Detecting Errors in Optical Music Recognition Output with Machine Learning

Timothy Raja de Reuse

Music Technology Area
Department of Music Research
Schulich School of Music
McGill University, Montreal
April 2024

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy

Contents

Abstract

Optical music recognition is the field of research that investigates how to computationally read musical documents. It offers powerful mechanisms for digitizing physical musical scores into machine-readable forms, enabling researchers to search, analyze, and interact with vast swathes of musical data in novel ways. However, a significant challenge encountered in leveraging this technology is the non-trivial errors it introduces into scores, especially on poor-quality scans or degraded historical musical documents. Manual correction of these errors, essential for analysis and archival, is a time-consuming task. I note that most of the errors these processes induce are "non-musical" in that they represent musical phenomena that a human composer would not have written. I propose that the laborious task of manual correction could be sped up by marking all symbols on a score that are musically unlikely, allowing the corrector to focus their attention only on regions of the score that contain errors, saving time in the process overall.

This dissertation covers the design, construction, and evaluation of a machine learning-based error detector for symbolic music with the objective of expediting the process of manual correction of optical music recognition outputs. This system is built around the Transformer network, chosen for its proven proficiency in capturing long-term dependencies. Given the high degree of repetition inherent to musical compositions, where themes and motifs may recur over long temporal distances, Transformer networks offer a robust way to detect anomalies or errors by learning from these repetitive characteristics.

As acquiring large quantities of annotated error data from musical scores is challenging, I devise several unique data augmentation methods to bolster the model's training dataset. These methods address the problem of data scarcity, improving the potential applicability of these methods in low-resource situations, where digitization via Optical Music Recognition stands to be especially helpful. I also develop a general and extensible definition of errors in symbolic music, based on sequence alignment algorithms, suitable for use in machine-learning applications.

This research aims to help bridge the gap between the promises of optical music recognition and its current limitations. Through intelligent error detection, it seeks to propel these technologies to their full potential as tools for musicological research and musical preservation.

Résumé

La reconnaissance optique de partitions musicales est un domaine de recherche qui se concentre sur la lecture automatique de documents musicaux. Elle propose des technologies pour transformer les images des partitions musicales en formats permettant des nouvelles manières d'analyse et d'interaction avec des grands volumes de données musicales. Cependant, ces technologies introduisent souvent des erreurs dans les partitions musicales pendant le processus de reconnaissance, surtout lors du traitement d'images de qualité médiocre. La correction manuelle de ces erreurs est une tâche longue et onéreuse, et constitue un obstacle à l'adoption de ces technologies. On remarque que la plupart de ces erreurs produisent des phénomènes musicaux qu'un compositeur humain n'aurait pas écrit. En conséquence, je propose que le processus laborieux de correction manuelle des erreurs pourrait être rendu plus efficace en marquant sur la partition musicale tous les symboles qui sont musicalement improbables. Cela permettrait aux relecteurs humains de se concentrer sur les secteurs de la partition qui contiennent des erreurs.

Cette thèse traite de la conception, de la construction et de l'évaluation d'un détecteur d'erreurs musicales, basé sur l'apprentissage automatique, dans le but d'accélérer la correction manuelle des résultats de processus de reconnaissance optique musicale. Le transformeur, un modèle d'apprentissage profond, est utilisé dans ce système pour capturer les schémas qui se répètent sur de longues périodes. Comme les compositions musicales présentent souvent un degré élevé de répétition, les thèmes et les motifs sont réitérés au cours d'une pièce, et les transformeurs seront capables de détecter des anomalies et des erreurs au cours d'un apprentissage automatique de ces caractéristiques répétitives.

Comme il est difficile de rassembler de grandes quantités de données d'erreurs annotées à partir de partitions musicales, je propose plusieurs nouvelles techniques d'augmentation et de renforcement de l'ensemble des données. Ces techniques améliorent l'applicabilité du détecteur d'erreurs dans des situations où la disponibilité des données de formation est limitée. Dans ces situations, la numérisation de partitions musicales sera la plus utile. Je propose également une définition générale et extensible des erreurs entre partitions musicales, basée sur l'alignement des séquences, et qui convient aux applications d'apprentissage automatique. Cette thèse est une exploration des limites actuelles de la reconnaissance optique musicale. Elle vise à augmenter le potentiel de ces technologies comme outils de recherche musicologique et de préservation musicale.

Acknowledgments

I would first like to thank my supervisor, Ichiro Fujinaga, who through weekly meetings and dozens of presentation run-throughs shared insights, refined my research ideas, and provided constant guidance to point me in the right direction. I am incredibly fortunate to have had such an attentive and supportive mentor over the seven years since I first came to Montreal.[1]

I thank my compatriots at the Distributed Digital Music Archives and Libraries Lab (DDMAL) who have worked alongside me over the last six years, including Gabriel Vigliensoni, Emily Hopkins, Alex Daigle, Junhao Wang, Finn Upham, Dylan Hillerbrand, Sevag Hanssian, Sylvain Margot, Yinan Zhou, Juliette Regimbal, Wanyi Lin, Jacob deGroot-Maggetti, Anna de Bakker, and Geneviève Gates-Panneton. The lab environment they created was friendly and warm, and always a pleasure to return to in the dead of the long Quebec winter. Special mention to Yaolong Ju, Martha Thomae, and Néstor Nápoles López, the other three doctoral students in DDMAL during my time there, for their collaboration, camaraderie, and commisseration. When I tell people I am a student of Music Technology and they inevitably ask "What does that mean, exactly?" I am always excited to describe the research we have done over the last five years.

I also thank the other members of the Computational Tonal Studies group[2] for working together with me on creating the Mendelssohn String Quartet Dataset. This experience, and the insights it gave me into how Optical Music Recognition fits into the larger digitization process, motivated the hypotheses that form the basis of this very dissertation.

I would also like to acknowlege members of the global Music Information Research community with whom I have collaborated on code, research, music, or ideas, including Antonio "Paco" Castellanos, Antonio Ríos-Vila, Elona Shatri, Laurent Feisthauer, Xaris Papaioannou, and Jonathan Orland. I doubt that any other research community is as welcoming, diverse, and full of people eager to synthesize knowledge together with people of other academic backgrounds.[3]

1. When I was presenting my first-ever conference paper at the International Society for Music Information Retrieval (ISMIR) conference in 2019, a man from a Korean telecommunications company walked up to my poster and started reading it, hands folded behind his back, without saying a word in greeting. Eventually his gaze drifted to the top of the poster, he saw the name of my supervisor as the co-author, and his eyes went wide. He said to me: "Ah, Fujinaga! You are a very fortunate guy." He then nodded once and walked off with a smile before I could respond. I think about this anecdote all the time.

2. Jacob deGroot-Maggetti, Laurent Feisthauer, Sam Howes, Yaolong Ju, Néstor Nápoles López, Suzuka Kokubu, Sylvain Margot, and Finn Upham.

3. I must acknowledge the international Stringology community, who I had the good fortune to meet at the Sequences in London workshop in 2023, and who also know how to have a good time.

Thanks to the McGill professors whose courses I took while a student, especially Phillipe Depalle, whose seminars on signal processing and time-frequency representations completely reformulated my understanding of how digital audio functions.

Thanks to the Centre for Interdisciplinary Research in Music Media and Technology (CIRMMT) for providing travel awards, research funding, and a forum for me to meet researchers with different backgrounds than my own. The research presented in this dissertation was further supported by the Fonds de recherche du Québec — Société et culture (FRQSC), who awarded me a doctoral scholarship from 2021–2023 (Dossier # 288312).

Finally, I thank my incredible partner, Grant Ongo, for his infinite patience and level-headed advice, and my parents, Shobhana Chelliah and Willem de Reuse,[4] for their eternal confidence and loud support. I'm excited to announce that you all never again have to ask me how my dissertation is going.

4. My father, Willem, also proofread the French translation of the abstract for this dissertation.

Contribution to Original Knowledge

- **Chapter 2:** This chapter offers a comprehensive survey of error detection and correction methods in real-valued sequences and Natural Language Processing (NLP), emphasizing their potential applications in symbolic music. As this is a relatively unexplored research area, the survey represents an original contribution to the field.

- **Chapter 3:** The chapter provides a detailed review of the field of Optical Music Recognition (OMR). Notably, it introduces novel methods for categorizing errors and presents an innovative approach (in Section 3.4) for estimating the potential time-saving benefits of an error detector.

- **Chapter 4:** This chapter surveys binary classification and machine learning, with a focus on the Transformer architecture. Section 4.4 discusses the challenges of learning long-term dependencies in music, constituting an original contribution.

- **Chapter 5:** The chapter describes a novel methodology for creating a machine learning-based error detector for symbolic music. It introduces an innovative agnostic encoding scheme for representing polyphonic, multi-staff music as sequences of one-dimensional categorical tokens. It presents a publicly accessible dataset of string quartets for training the error detector and the development of a novel difference operation on musical sequences, leveraging the Affine Needleman-Wunsch (ANW) algorithm. This chapter also outlines a machine learning pipeline for training, testing, and inference, tailored for the developed methodology.

- **Chapter 6:** This chapter details the experimental evaluation of the error detector. It includes comprehensive trials that assess the impacts of various architectural choices, sequence length variations, data augmentation methods, and fine-tuning strategies on the model's performance.

Contribution of Authors

- **Chapter 1:** All contributions in this chapter are mine.
- **Chapter 2:** All contributions in this chapter are mine.
- **Chapter 3:** The discussion of OMR errors presented in this chapter takes inspiration from the conference paper covering the creation of the Mendelssohn String Quartet Dataset (MSQD), which describes the process of correction OMR outputs in detail (deGroot-Maggetti et al. 2020).
- **Chapter 4:** All contributions in this chapter are mine.
- **Chapter 5:** The methodology presented here is adapted from a previous conference paper detailing an earlier version of this error detector (de Reuse and Fujinaga 2022).
- **Chapter 6:** All contributions in this chapter are mine.
- **Chapter 7:** All contributions in this chapter are mine.

The large language model-based chatbot ChatGPT (Chat Generative Pre-trained Transformer), versions 3.5 and 4, was used throughout this dissertation to proofread my written prose and rephrase certain sections for clarity. It was also used to generate LaTeXmarkup code for some figures and tables.

Abbreviations

ABC Annotated Beethoven Corpus
AMT Automatic Music Transcription
ANW Affine Needleman-Wunsch
AP Average Precision
API Application Programming Interface
CWMN Common Western Music Notation
FNN Feed-forward Neural Network
GEC Grammar Error Correction
GED Grammar Error Detection
GPU Graphics Processing Unit
GUI Graphical User Interface
HRIR High-Recall Information Retrieval
IMSLP International Music Score Library Project
KLM Keystroke-Level Model
KNN K-Nearest Neighbors
LSTM Long Short-Term Memory
LSTUT Long Short-Term Universal Transformer
MCC Matthews Correlation Coefficient
MEI Music Encoding Initiative
MIR Music Information Retrieval
MIDI Musical Instrument Digital Interface
MSQD Mendelssohn String Quartet Dataset
NLP Natural Language Processing
NW Needleman-Wunsch
OCR Optical Character Recognition
OMR Optical Music Recognition
OSSQ OpenScore String Quartets
PDF Portable Document Format
QKV Query-Key-Value
RNN Recurrent Neural Network
SGD Stochiastic Gradient Descent
UT Universal Transformer
XML Extensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation describes a machine-learning method for detecting incorrect notes in transcribed musical scores, intended to speed up the tedious task of human review and correction of Optical Music Recognition (OMR) processes. In other words, it is a kind of musical spellchecker that allows the corrector to focus their attention only on regions of the score that have errors.

This chapter contains introductory material to illustrate the problems I am trying to solve, and the rationale behind the methods I use to solve them. Section 1.1 discusses the reasons why turning images of scores into machine-readable formats is an important task in the first place, and why OMR technologies are useful for that task. Section 1.2 describes intuitions for why a musical error detector might be possible, initial thoughts on how it might be designed to be most useful to human correctors, and several challenges that must be addressed in designing the method. Section 1.4 outlines the remainder of the dissertation.

1.1 Why is Symbolic Music Important?

Musical scores are, increasingly, becoming digital objects. This is a recent development. For much of the history of Western music, even after the maturity of the music publishing industry, in order to play or analyze a particular musical score it was necessary to seek out a physical copy of it, sometimes at great effort and expense, assuming one could be located at all. Today, vast digital archives like the publicly accessible International Music Score Library Project (IMSLP) host over 730,000 scores in image formats available for free to the general public.[1] This collection continues to expand as researchers, archivists, and enthusiasts digitize and upload more scores.

1. `imslp.org`

However, not all forms of digitization are equally valuable. In this section I argue for the importance of *symbolic music* as an archival format alongside digital images, motivating the development of technologies to speed up the process of encoding images of scores into symbolic formats.

Symbolic music refers to music represented in a machine-readable structured format, typically as a list of events that occur at particular times. Examples of symbolic formats include Humdrum **kern, Musical Instrument Digital Interface (MIDI), and formats based on Extensible Markup Language (XML), like MusicXML and Music Encoding Initiative (MEI). When I speak of scores in symbolic format as *machine-readable*, this refers specifically to their semantic content: the notes, rests, events, instruments, and other musical information. It is often contrasted with audio, which can be played back but does not include semantic musical information. "Digitization" can sometimes refer only to the scanning of musical document into digital image files, that, while digital, are not machine-readable in the same way. Digital images of scores are more easily distributed and accessible than physical documents, but computers cannot access musical information like the number of parts, the types of instruments, or any musical content from just an image file. This dissertation is primarily concerned with the process of turning extant images of scores into symbolic music files, and not with the process of capturing those images from physical sources.

Large-scale encoding of musical works into machine-readable formats provides a number of benefits over simply digitizing them into images. Performers of music can automatically generate part scores or transpositions of works for which such things were never published. Music theorists and musicologists have access to an interactive version of the musical surface that can be annotated, edited, and pieced apart in one of many notation editors, providing an invaluable supplement to printed scores and recordings that allows for deeper analysis. Detailed musical information offers new ways to structure and search databases of scores, enabling otherwise impossible tasks like search-by-melody. The burgeoning field of computational musicology finds ways to use computer tools to automatically analyze huge quantities of symbolic music at once. New types of questions in music research, about the stylistic tendencies of huge corpora of music spanning centuries, would have been totally unanswerable without large collections of symbolic music. Finally, when music previously only available in physical copy or in scanned images is made available in symbolic format, it is more easily disseminated and more accessible to the general public. Digitization into a data-rich symbolic format is thus an important tool for

preserving objects of important cultural heritage that might otherwise be forgotten (Thomae 2023).

A significant amount of music currently exists solely as non-machine-readable digital images. Manual *transcription*, where a skilled music engraver manually inputs the content of a digital score images into a notation editor, is a slow and expensive process that does not scale well to situations where hundreds or thousands of documents must be encoded into symbolic formats. The field of OMR is dedicated to the development of technologies to automatically analyze and transform musical documents into machine-readable forms, analogous to the field of Optical Character Recognition (OCR) for text. Typically, the primary objective of OMR is to convert scanned images of musical scores into symbolic music files, although there are other applications (discussed in Section 3.1). Despite its potential as a crucial tool for efficient digitization of extensive musical archives, OMR technology has not yet reached a level of reliability sufficient for large-scale applications. Hankinson (2015) observed that all existing commercial OMR software is tailored for personal use and lacks the necessary features for digitizing music on a large scale. This situation remains unchanged in recent years; even the most developed commercial OMR applications lack the functionality to batch-process collections of musical documents without manual human intervention on each one.

The primary obstacle to the universal application of OMR in large-scale symbolic encoding of musical works is the significant error rate in the outputs of existing techniques. To make the outputs suitable for musicians or music researchers, they require manual correction. Like manual transcription, this is a tedious and time-intensive task that forms a bottleneck in the digitization process. Correcting requires expertise in music transcription, since efficient score editing demands both musical knowledge and experience with music notation software. Although specialized interfaces have been developed to aid human correction (Rizo et al. 2018), the process remains tedious and expensive. Usually, it involves meticulously comparing the OMR results with the original scan for every measure in a repetitive sequence to ensure complete accuracy. OMR can speed up the digitization process under certain conditions (Alfaro-Contreras et al. 2021). However, for complex polyphonic scores, the combined steps of scanning, performing recognition, and subsequent corrections can sometimes be more time-intensive than manual transcription a physical score into a symbolic format (Daigle 2020).

1.2 Motivation and Hypotheses

The concept for this error detector arose from collaborating on a study that revolved around correcting OMR output (deGroot-Maggetti et al. 2020), in which I experienced firsthand the tedious and time-consuming process of manual error correction in music. Its design was further motivated by my own previous research into regularity and repetition in music.

During my correction of OMR output of several string quartet movements, I observed that the mistakes introduced into the symbolic music were highly "unmusical." By this, I mean that they were musical phenomena that, by my knowledge of the conventions of Romantic string quartets, would almost surely never be written intentionally. These included obvious errors like measures containing what amounted to musical gibberish, which occurred when dense runs of sixteenth notes were obscured by a smudge or imperfection in the scan and cause the OMR system to be totally unable to parse the page. More subtle were disruptions in recurring articulation patterns, missing dynamic markings on one out of several staves, or the misidentification of an accidental as a notehead leading to a dissonant cluster chord. Additionally, certain rhythm anomalies or syncopations, while not entirely absent in Romantic compositions, were identifiable as errors by the way they seemed distinctly out of place within their musical context. Often, these occured when a single note's duration was read incorrectly, and the onset times of other events in the same measure were shifted to compensate.

This correction process led me to hypothesize that a machine-learning model could potentially detect a majority of the errors generated by an OMR system without having access to the original score in any form. This would necessitate embedding into the model an understanding of how musical notation is typically used as well as the specific conventions of the piece's genre. In other words, the model would need to replicate the musical intuition that enabled me to recognize rhythms which appeared "off" within their context for reasons I could not articulate. If effective, this error detector could substantially reduce correction time. Rather than comparing every OMR output measure to its counterpart in the original score, editors would only inspect areas the algorithm identifies as potentially flawed. Time would be saved even if the model occasionally flags many non-faulty regions of the score, provided it correctly excludes significant portions from requiring manual review. For instance, if the algorithm detects all errors while marking 50% of the score, this indicates that half the score does not require human examination. Similar approaches are already employed in the domains of Grammar Error De-

tection (GED) and Grammar Error Correction (GEC), which focus on identifying and rectifying errors in human language, as discussed in Section 2.2.

In developing the model for this research, I propose two additional hypotheses related to its design. Firstly, considering the vast amount of training data used in similar machine-learning applications in other domains, which far exceeds what is typically available for symbolic music tasks, I hypothesize that data augmentation techniques, akin to those employed in GEC and GED, can be effectively used to mitigate the challenge of data scarcity in symbolic music. Secondly, I recognize that most musical genres exhibit a high degree of repetitiveness, presenting a form of redundancy that could be exploited in error detection. I posit that if the model can analyze and learn from long-term dependencies in musical data, it should leverage this repetitiveness to enhance its error detection capabilities. For instance, if an OMR process corrupts a recurring melodic figure in a few instances, the model might identify these errors by noting deviations from the established repetitive pattern. Consequently, I hypothesize that extending the sequence lengths fed into the model, providing it with access to a broader span of musical context, will improve its accuracy in identifying errors.

1.2.1 Is Error Detection in Music Possible?

The feasibility of this task may seem uncertain, given the nature of musical rules compared to the syntax of natural languages. Natural languages, such as English, have well-defined syntax rules that dictate the placement and modification of words to maintain grammatical correctness. In most cases, sentences can be unambiguously assigned into the classes of "grammatically correct" or "grammatically incorrect."[2] In contrast, syntactic rules of music, especially regarding genre conventions, are more tendencies than rigid, inviolable rules. Unlike the more concrete syntax rules observed in formal registers of natural languages, musical rules are less explicitly defined and more flexible.

I would argue that error detection in music is still possible because music is unusually repetitious from an informational standpoint. This makes music unique among all forms of art created by humans; furthermore, repetitiousness could be argued to be the one thing that *all* musical practices have in common across all human cultures (Margulis 2014, 4–7). Numerous theorists and composers have observed this characteristic. Johannes Brahms, a Romantic composer, ex-

2. This is a considerable oversimplification. Compared to the rules that govern natural languages written in a formal register, however, the rules of music are still much less concretely defined.

tensively employed developing variation in his compositions, repeatedly presenting single themes that undergo gradual alterations and reinterpretations, always anchored back to a core motif (Frisch 1982). Arnold Schoenberg's compositional approach revolved around the concept of the *Grundgestalt*, or *basic shape*, where he crafted entire pieces centered on exploration of a single dynamic, melodic, harmonic, or textural idea (Lamont and Dibben 2001). While 20th-century popular music is frequently cited as being highly repetitive relative to classical compositions, Middleton (1983) contends that Western art music prior to the 1900s exhibited analogous levels of repetition as contemporary popular music within individual works. These two genres simply used repetition in different ways: they repeat on different time-frames, using different techniques, and in different aspects of the musical surface. Indeed, all genres of music have their own tendencies not only in terms of melody, harmony, and rhythm, but in terms of what types of structure and repetition are encouraged. I propose the following analogy: an individual work of music builds through repetition its own "vocabulary" and "syntax," and in order to be coherent to the listener, the work must obey these self-imposed internal rules. As a result, it must have a certain level of informational redundancy, and perhaps this redundancy can be exploited.

I posit that despite the fluidity and variability in musical rules compared to the rigid structure of grammar and syntax in language, it is feasible to algorithmically detect "wrong notes" in musical scores. This detection should rely not solely on adherence to genre conventions but also on intra-piece comparisons. In natural language, a grammatically incorrect sentence is incorrect whether it appears in an encyclopedia or in a magazine; by contrast, musical elements can be deemed appropriate or erroneous depending on their context within a piece. For example, a complex, dissonant chord might be typical in a Wagner opera but an obvious mistake of transcription in a Bach chorale. The repetitive nature of musical compositions could be particularly instrumental in identifying errors in OMR outputs. An irregular chord might be correctly identified based on its recurrence elsewhere in the piece or marked as an error if it disrupts a consistently repeated motif. This method might not be effective for genres lacking predictability and repetition, such as late twentieth-century art music or transcriptions of contemporary jazz solos.

Choosing machine learning to develop the error detector seems the most viable approach. The intricate and often ambiguous nature of musical rules makes manual coding of genre conventions both impractical and too restrictive, capturing only a fraction of potential OMR errors. Furthermore, rules would need constant adaptation for each new genre. In contrast, machine

learning offers the potential for a genre-agnostic error detection system, applicable across various musical repertoires. The success of machine-learning models in mastering complex human grammar and detecting and correcting textual errors suggests a promising avenue for learning and replicating musical structures, especially with appropriate training data. Deep learning-based musical sequence models have already demonstrated potential in capturing and replicating specific musical idioms (Ji et al. 2023).

1.2.2 Why Detection and not Correction?

The reader may further question why I aim for *detection* of errors as opposed to *correction* of errors outright. Automatically fixing some or all of the errors in a piece could reduce the time needed to edit OMR output even more than simply marking detected errors. There exists a large body of work on error correction for natural language that could be drawn on for this purpose. A reasonable hypothesis supporting this approach might be that in sufficiently repetitive music, deviations from that repetitive structure can not only be detected but also corrected by ensuring that the repetitive structure is maintained. However, in music, correction is a problem orders of magnitude more difficult than detection.

When I began this research project, error correction in OMR output was my intended goal. I discovered that it is possible to train a machine-learning model to replace errors with something that is musically sensible, and would not sound "wrong" if performed. It is much more difficult, however, to get it to exactly match the decisions of the original composer. My attempts at constructing error correction models would often inadvertently add just as many errors to the scores as they successfully corrected. There only exist a handful of published studies on correcting errors in polyphonic music (discussed in Section 2.5) and most of them also achieve a level of performance not high enough to be useful for correcting OMR output, although some do perform better than random chance.

Regardless, I posit that unless a correction system is capable of correcting *nearly all* errors in OMR output, it would not significantly save time on the part of the human corrector. An imperfect correction would still require the human transcriber to manually review the remainder of the output. In the end, I opt for error detection because a highly performant error detector is probably more useful a tool than an error corrector of lesser quality.

1.3 Challenges and Sub-problems

I identify three challenges that I must solve in order to construct a machine-learning model to perform the task of musical error detection: the availability of data to train the error detection models (Section 1.3.1), the representation of the musical score and the accompanying definition of an "error" on that representation (Section 1.3.2), and the method of evaluating the results (Section 1.3.3).

1.3.1 Overcoming Data Scarcity

In order to create a machine-learning model to train an error detector, it is necessary to collect *training data* in the form of examples of the types of errors that the model will detect. Creating training data by introducing random errors into existing correct scores and then tasking the detector to identify them would not result in a system that generalizes well to actual OMR outputs. Errors introduced by OMR processes are not distributed uniformly within musical scores; certain musical features tend to be more susceptible to transcription errors. For example, in dense passages smaller symbols like articulation markings and accidentals are often missing or misinterpreted. Conversely, quarter notes, half notes, and whole notes within less cluttered sections tend to be accurately identified. The manual correction process benefits from the editor's awareness of the OMR system's idiosyncrasies and its usual pitfalls, allowing for a more efficient review and rectification. Ideally, an algorithmic error detector should also possess such a familiarity with the OMR system whose errors it corrects.

To enable a machine-learning model to pinpoint these errors, it should be trained on a dataset containing real OMR errors paired with their corrected counterparts. However, there are few existing datasets tailored for this purpose. Current datasets geared towards OMR research prioritize the creation of ground truth for OMR techniques to be trained and tested on, rather than the analysis of these algorithms' outputs. Given that many commercial OMR systems are created for personal use, they lack provisions for batch processing, and so the automated construction of an appropriate dataset would be a challenge. A potential technique for overcoming this involves manipulating error-free datasets by deliberately injecting them with errors statistically similar to those found in OMR outputs. This process, called *data augmentation*, is often used in analogous error detection and correction tasks on natural language (See Section 2.2.1).

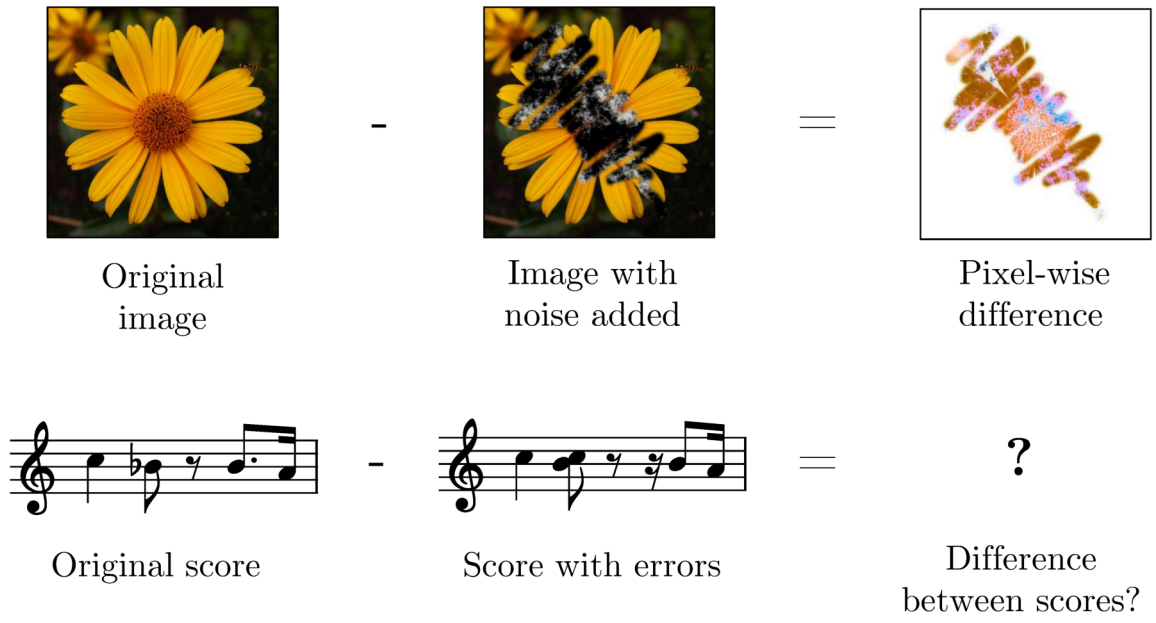


Figure 1.1: An visual analogy illustrating why it is nontrivial to define the notion of an "error" on a musical score.

1.3.2 Error Representation

Representing musical notation as symbolic music without losing information on all the ways that symbols can be arranged on the page is difficult. In polyphonic music, multiple notes can sound simultaneously on a single staff, and notes can begin or end while other notes are still sounding. Even more complicated are single objects that span durations in time like slurs, ties, and crescendo / diminuendo markings. Symbolic music formats like MusicXML and MEI represent music as hierarchical trees, where symbols on the page are grouped together and some symbols naturally "belong" to others (e.g., a dot of augmentation belongs to the note it applies to). In order to build a tool that can detect errors in musical scores, it is necessary to define what an error on a musical score *is*, and the inherently hierarchical nature of music notation makes this difficult.

Figure 1.1 illustrates this problem by way of analogy with the domain of digital raster images. Defining the error between two images is straightforward. In data, images are simply two-dimensional arrays of pixels, with each pixel (typically) three numerical values, one each for the blue, red and green channel. If one image is a version of another with errors added, then one can define the error by taking the pixel-wise difference between each pixel of each channel in both images, and this would work no matter the shape, nature, or intensity of the error that was introduced to the image. A hypothetical error detector that operates on images could use

this simple scheme to define its operations like so: *Given an image containing errors, this error detector finds the pixels whose values would need to change in order to correct the image.* For music, there is no such obvious difference operation. Asking "How many differences are there between these two scores?" of the two short related excerpts in Figure 1.1 may prompt the following questions:

- If the onset time of a note appears to change but its notated duration and pitch are the same, then is it the same note? Is that note erroneous?
- If a note's accidental is removed, is the note fundamentally changed? Is the note itself erroneous in this situation, or is it just the removal of the accidental that constitutes an error?
- Is a note changed when an accidental on the same pitch in the same measure is removed, thus changing its pitch from afar, even if it visually remains the same?
- Is a rest an object in itself, or is it (like in MIDI files) just a placeholder that represents time when no notes are sounding? Does it make sense to talk of a rest being "deleted" when a note fills its space, or "inserted" when a note's duration is shortened?

A machine-learning model designed for error detection in musical scores demands not just a generic means of discerning differences between scores, but a method tailored to a specific machine-learning framework. There do exist methods of taking differences between scores that take into account all parts of musical notation (see Section 2.5.2), but the outputs of these methods tend to be complicated, human-readable objects that are themselves hierarchically organized. This is an issue because on a high level, machine-learning models require inputs of a uniform shape, both during training and inference (This is discussed more formally in Chapter 4). If a operation to take differences between scores yields results of varying shape depending on the input scores' length and attributes, as is the case with most existing methods of taking differences between scores, then designing a machine-learning architecture to train on those errors is especially difficult.

1.3.3 Difficulty of Evaluation

To reiterate a primary objective of the error detector: if it can confidently designate half of a score as erroneous, the remaining half can be disregarded for human review, provided that the

unflagged sections of the score have an almost negligible rate of missed errors. This categorizes the error detection task as a *high-recall* problem where ensuring that all errors are identified is crucial, even if it means an increase in the number of false positives. Metrics tailored for such tasks are necessary to gauge performance accurately, and careful interpretation of results is imperative, as discussed in Section 4.1.3. The kinds of standard numerical scores pertinent to other retrieval tasks, such as the F1 score commonly used to evaluate binary classification tasks, may not meaningfully represent the model's usefulness in an interactive correction scenario.

The intricate nature of musical notation makes the assessment of OMR algorithms challenging, as expanded upon in Section 3.2. Many of these intricacies are also pertinent when evaluating the music error detector. The chosen metrics for evaluation should align with the properties of the selected error representation method. Since the goal of the error detector is to reduce the total time necessary for human correction, in Section 3.4 I devise a method, based on the Keystroke-Level Model (KLM) of user interaction, for estimating the potential time savings of the error detector. While imprecise, this estimation gives a rough idea of the potential utility of the error detector, and can evaluate under what conditions the error detector can be expected to save a significant amount of time.

1.4 Dissertation Outline

This dissertation comprises seven chapters. Chapter 1 provides a brief outline of the applications of OMR, the reasons an error detector would be beneficial for large-scale music digitization workflows, and the main challenges presented by this undertaking.

Chapter 2 discusses the concept of errors and outliers in depth. This chapter reviews definitions of errors and outliers that have been used in various fields, methods for finding errors in both real-valued sequences and natural language, and the relatively small amount of research on error analysis in symbolic music.

Chapter 3 covers the field of OMR, its history, the most common methods used, mechanisms by which errors manifest in the OMR process, and the difficult task of defining sensible, interpretable evaluation metrics for OMR methods. I also discuss the commercial PhotoScore OMR application, which will be used to generate OMR errors for the method to detect.

Chapter 4 reviews the task of binary classification (a category including error detection) and the many existing metrics for evaluating the results of binary classifiers. I provide an overview of

the field of machine learning, building from base principles up to modern attention mechanisms.

Chapter 5 presents the methodology used to design the error detector. This covers the particular symbolic encoding of musical scores used, the way that errors are defined in this encoding method, the dataset gathered, data augmentation methods, and the machine-learning pipelines used for training, testing, and inference.

Chapter 6 describes the design of the experiments to evaluate the error detector, and presents the results of these experiments along with a variety of performance metrics and discussion of how to interpret these numerical scores. I supplement these quantitative results with several long-form examples of musical scores, containing errors from an OMR process, annotated with the predictions of the error detector. I then discuss how the results reflect on the detector's potential usability in actual OMR correction scenarios.

Finally, Chapter 7 provides an overview of the contributions of this research. I discuss the usability of the error detector and whether I have achieved the design goals set out, the results of the hypotheses made in this chapter, describe possible future directions, and provide information on reproducibility and dataset availability.

Chapter 2

Errors and Outliers in Sequence Data

To motivate the design of the musical error detector, it is necessary that I first cover how error detection works in other sequences. In this chapter, I review error detection within sequences, a task variously referred to as "error detection," "outlier detection," "anomaly detection," and "novelty detection." In line with common terminology in literature, I use "outlier" for real-valued datasets and sequences, and "error" for categorical data. I also cover methods for taking differences between sequences, and the relationship of these methods to outlier analysis. The tools described here will be used in Chapter 5 to define a notion of "error" in symbolic music suitable for input into a machine-learning model.

The goal of this chapter is to establish a broad and comprehensive understanding of the concept of an "outlier" in a dataset. Hawkins (1980) defined an outlier as "an observation that significantly deviates from others to the extent that suspicions arise about its generation through a distinct mechanism." This leads to an analogy involving two mechanisms: a *primary mechanism* responsible for generating the majority of the data, and a *secondary mechanism* that occasionally interferes, creating anomalies. Error detection in datasets, therefore, involves understanding and modeling the primary mechanism. This model is then used to identify instances where data diverges from the expected pattern, indicating potential errors or outliers. This approach necessitates a close analysis of the data generation process in order to create a model (explicit or implicit) of the primary mechanism to effectively identify deviations that represent errors or outliers.

First, in Section 2.1, I review outlier detection methods in their broadest sense, including the use of simple statistical tests, and then focus on methods that find outliers on continuously varying real-valued signals, including both non-parametric and model-based techniques.

Moving forward, in Section 2.2 I focus on methods derived from the domain of Natural Language Processing (NLP), specifically those geared towards identifying errors within written text: the fields of Grammar Error Detection (GED) and Grammar Error Correction (GEC). While error analysis in natural language falls within the larger umbrella of outlier detection methods, the fields under the umbrella of NLP employ a distinct set of algorithms and underlying assumptions compared to outlier detection approaches for real-valued signals. Moreover, parallels exist between the realms of natural language and symbolic music, prompting the application of NLP techniques in music-related contexts. This is particularly justified considering the substantial volume of literature on error analysis within NLP compared to the much smaller volume that exists for musical contexts. In particular, Section 2.2.1 discusses methods for data augmentation in the fields of GEC and GED that will be adapted in Chapter 5 to bolster the size of the musical datasets used for training the error detector.

Section 2.3 discusses the differences between supervised outlier detection, where labeled examples of outliers are available, and unsupervised detection, where labeled examples are not available. These two fields require vastly different approaches to both their algorithmic implementations and the interpretation of their results.

In Section 2.4 I present common algorithms for taking differences between sequences, starting from measures of edit distance. I will discuss the concept of a sequence alignment, how alignments are computed, the computational complexity of standard methods, and how such alignments may be interpreted to provide definitions of errors in sequences.

Lastly, Section 2.5 reviews applications of the concepts of the previous sections on musical data. I will cover the work that has been done on error detection and analysis of errors in music, along with related work on using edit distances and sequence alignments for music retrieval and evaluation of music retrieval tasks.

2.1 Outliers in Real-valued Sets and Sequences

This section discusses outlier detection in collections of real-valued data. A real-valued sequence is an ordered list of elements, each holding a real numerical value, distinguishing it from categorical values used in fields like NLP. These sequences may be *univariate*, comprising single real values, or *multivariate*, where elements are n-tuples of real values for $n > 1$. While many summarized approaches target *time-series* data, characterized by regular interval observations

over time, this discussion adopts the term *sequence* to encompass both time-indexed and other data types. An example of a non-time-series sequence, indexed by geographic position rather than time, might be "the value in dollars of every property along a given long road through a city, ordered from east to west, on a given day."

Outlier detection in sequential data differs from unordered datasets, where each data point is independent. Types of outliers in sequences vary: point-outliers are single elements, subsequence outliers span fixed or variable ranges, and entire sequences may be outliers within larger datasets. These can involve either univariate or multivariate sequences. Detection methods can be univariate, focusing on a single variable in a multivariate series, or multivariate, considering multiple variables.

The initial discussion of this section covers simpler techniques for non-sequence data to provide context, as many sequence error detection methods are extensions of these. According to the taxonomy developed by Blázquez-García et al. (2022), outlier detection in sequences divides into two categories: *non-parametric* (or *instance-based*) methods, which require no training phase and make no assumptions about data distribution, and *model-based* methods, which use statistical or machine-learning models for data analysis.

2.1.1 Outlier Detection in Unordered Data

The simplest methods of finding outliers in data assume datasets to be identically and independently distributed, which is not the case for most sequence data. These tests can still be used to find outliers in sequence data by ignoring the sequential aspect of the data.

The Student's t-test, introduced by Student (1908), is a foundational method for detecting outliers in univariate data. It operates by fitting a statistical model to the data and identifying data points as outliers if they are improbably generated by this model, as determined by a specified confidence level. For different distributions, analogous tests are available. Ferguson (1961), for instance, proposed a method to detect outliers in normally distributed datasets, assuming outliers originate from a normal distribution with differing mean or variance.

For multivariate data, the Mahalanobis distance, as described by Mahalanobis (1936), is commonly used. This metric measures the distance of each data point from the mean, adjusting for the variance in all dimensions, effectively transforming the dataset to have a unit variance. This operation presupposes that the data comes from a multivariate normal distribution, typically visualized as an ellipsoid in $\mathbb{R}^n$. However, this method can be influenced by outliers themselves,

which may distort the calculated mean and covariance matrix, leading to inaccurate detection of outliers. This phenomenon is known as "masking," and to address it, Penny (1996) analyzed a variation called the jackknifed Mahalanobis distance, which calculates these metrics using only a portion of the dataset to avoid incorporating outliers.

Detecting outliers in multivariate data that does not conform to a multivariate normal distribution is more challenging. In such cases, the individual values of a data point may fall within expected ranges for each dimension, yet the point may still be an outlier. Approaches in this area involve estimating the covariance matrix and using it to reduce the dimensionality of the data, often through methods like principal component analysis, as discussed by Caussinus and Ruiz (1990). A comprehensive review of various robust statistical techniques for detecting outliers in non-normally distributed multivariate data was provided by Ben-Gal (2005).

2.1.2 Non-parametric Sequence Outlier Detection

Non-parametric outlier detection in sequence data falls into two primary categories, as delineated by Blázquez-García et al. (2022): *histogram-based* (also known as *deviant-based*) methods and *density-based* (also known as *distance-* or *dissimilarity-based*) methods.

Histogram-based methods rest on the notion that outliers influence the summary statistics of a dataset more than non-outliers do. For example, if the mean or standard deviation of a dataset changes significantly when a particular data point is excluded from the dataset, then that data point is likely an outlier. Jagadish et al. (1999) developed an outlier detection approach that starts by creating a histogram representation[1] of the sequence, then downsampling it irregularly to minimize deviation from the original sequence. The key step involves assessing the impact on reconstruction error if a particular point is removed; a substantial decrease in error signifies the point as an outlier, as it is divergent from its surroundings. Subramaniam et al. (2008) extended this concept to environmental sensor networks with limited computational resources, focusing on standard deviation within each histogram bin rather than raw magnitudes.

Density-based methods, conversely, draw on principles akin to nearest-neighbors classification (Cover and Hart 1967). In this framework, a data point is deemed an outlier if it has fewer than k neighbors within a specified radius R. Another variant defines a point as an outlier if the aggregate distance to its k nearest neighbors exceeds a certain threshold (Angiulli and

1. This downsampling process is distinct from the more common use of the term "histogram", where it refers to a method of visually representing the distribution of a collection of real-valued data that is not necessarily ordered.

Pizzuti 2005). When k is set to the dataset's size, this method converges to the Mahalanobis distance-based approach for unordered multivariate data (Section 2.1.1). To adapt this to ordered data, Angiulli and Fassetti (2007) suggested evaluating each data point for outlier status using nearest-neighbor methodology and a selection of sequence elements in proximate indices. This defines outliers as points significantly dissimilar to their sequential neighbors, disregarding other sequential relationships.

Yu et al. (2014) introduced the concept of a *nearest-neighbor window* to handle univariate sequence outlier detection by converting it into a problem of detecting outliers in multivariate unordered data. In their approach, for any point p_i in a univariate sequence, the k-nearest-neighbor window, denoted as η_i, is defined as a $2k$-tuple containing the points surrounding p_i:

$$\eta_i = (p_{i-k}, p_{i-k+1}, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{i+k-1}, p_{i+k}). \tag{2.1}$$

This tuple represents the temporal context of p_i in a $2k$-dimensional space. To identify outliers, methods like nearest-neighbor searches or Mahalanobis distance-based detection are applied to this dataset of nearest-neighbor windows. The key assumption is that points with similar contexts should be similar; if a point significantly differs from others with similar contexts, it is considered an outlier.

Both histogram- and density-based methods primarily address univariate sequences. Extending these to multivariate sequences often involves dimensionality reduction, transforming a complex sequence into fewer uncorrelated sequences on which separate outlier analyses can be performed (Papadimitriou et al. 2005). A limitation of these methods is their indirect use of temporal information. Histogram-based approaches consider the entire sequence but lack the capability to make decisions based on the local order of elements. Density-based methods can be more effective, contingent on their implementation of nearest-neighbor searches. The approach by Yu et al. (2014), while ordering nearest-neighbor windows, struggles to detect outliers resulting from insertions or deletions in sequences. Hence, these techniques are best suited for situations where outliers are discernible solely by magnitude in relation to their surroundings, but they falter in identifying outliers rooted in disruptions of repetitive patterns.

To address outliers in sequences with repetitive behavior, one strategy involves preprocessing the sequence with a transformation, like the Fourier transform, that accentuates or clarifies its repetitive nature. Outliers can then be detected in this transformed sequence using simpler

non-parametric or statistical methods (Erkuş and Purutçuoğlu 2021).

2.1.3 Model-Based Sequence Error Detection

Outlier detection can be approached using model-based methods, which involve constructing a predictive model of a dataset's primary mechanism. These models predict the expected behavior of data points based on their context and identify significant deviations as outliers. Some reviews of the literature, such as that of Blázquez-García et al. (2022), separate this class further into *prediction models*, which attempt to predict data points of a time series using only previous data points, and *estimation models*, which are not restricted in this way. I make no such distinction here, since when performing Optical Music Recognition (OMR) one generally performs it on an entire page or piece of music at once. I will refer to all models capable of making predictions about sequence data as *predictive models* to better match terminology used in machine learning.

Predictive models can be constructed using a wide variety of stategies. Some methods assume each data point to be close to the median of its neighbors (Basu and Meckesheimer 2007), while others might employ a Markov model for predicting future time steps (Ozkan et al. 2016). Advanced machine-learning techniques are also widely used in this domain: Malhotra et al. (2015) leveraged Long Short-Term Memory (LSTM), Aguayo and Barreto (2017) utilized self-organizing neural maps, and Munir et al. (2019) employed deep convolutional neural networks. For a detailed list of machine-learning techniques in outlier detection, see Chalapathy and Chawla (2019) and also Chapter 4 of this thesis for a broader overview of machine-learning methods.

Autoencoders, introduced by Hinton and Salakhutdinov (2006), are particularly useful for error identification and removal. They function by encoding objects (here, time-series data) into a lower-dimensional latent space, then decoding them back to their original form, minimizing the loss between the original and the reconstruction. This process effectively learns a dataset-specific form of lossy compression, which inherently requires robustness to noise. This makes autoencoders valuable in applications like image deblurring or denoising (Bigdeli and Zwicker 2017). For outlier detection, the processed time series from an autoencoder is compared with the original; differences indicate areas where the autoencoder has filtered out "noise", potentially highlighting outliers. Xia et al. (2015) first demonstrated this approach. The versatility of autoencoders allows for a range of underlying machine-learning architectures, including recurrent (Kieu et al. 2019), convolutional (Kieu et al. 2018), and simple feed-forward networks (Sakurada and Yairi 2014). A key advantage of autoencoders in this domain is their capacity for unsuper-

vised learning, which allows them to function without needing labeled examples of outliers (see Section 2.3).

2.2 Grammar Error Correction and Detection

Of all fields that approach error and outlier detection, Grammar Error Correction (GEC) and Grammar Error Detection (GED) are the most directly related to the task of detecting errors in OMR output. The primary goal of GEC is to input text with grammatical errors or misspellings and output a corrected version, typically formulated as a sequence-to-sequence model akin to machine translation. In contrast, GED focuses on classifying elements of an input sequence as either erroneous or correct, without necessarily correcting the errors. This can be approached as a binary classification or a multi-class classification problem, with the latter also identifying error types. These fields have evolved from early rule-based systems to more sophisticated approaches, using machine-learning techniques from other areas of the field of NLP.

The Writer's Workbench (Macdonald et al. 1982) was an early, widely used suite of Unix command-line tools for grammar evaluation. It consisted of numerous programs, each focusing on specific grammatical or stylistic rules. Programs like `double` detected repeated words, while `gram` identified simple grammatical mistakes. These tools largely relied on string matching and replacement against databases of common errors. For example, `diction` (Cherry and Vesterman 1981) searched for phrases from a database of 450 common grammatical issues. Naber (2003) developed a rule-based English grammar checker effective for common errors made by English learners, but each error type required manual encoding. Some grammatical errors were challenging to encode in this system.

Rule-based systems, while effective for certain error types, have limitations in their adaptability and coverage. As language and usage evolve, these systems require constant updating and expansion to remain effective. Additionally, they may not efficiently handle complex or nuanced errors, which are better addressed by more advanced machine-learning techniques.

Syntax-based systems, which are more robust than rule-based systems, perform syntactic analysis of input text through part-of-speech tagging and parsing of sentences into syntactic tree structures. However, these systems face challenges due to the inherent ambiguities in natural languages, where multiple valid parse trees for sentences are possible, complicating the definition of good performance. Ranjan et al. (2016) provides a comprehensive review of part-of-speech

tagging and automatic syntactic parsing.

While more powerful, syntax-based methods often lack the interpretability of rule-based systems. For example, a rule-based grammar checker can explicitly indicate the broken rule causing an error, whereas a syntax-based method might only indicate a failure to parse. This issue is particularly pronounced in languages like Arabic with freer word order, as noted by Shaalan (2005). Their approach combined syntax-based parsing with additional rule-based analysis for sentences that failed initial parsing. Other strategies include relaxing parsing rules to find approximate valid parses, assigning costs to ungrammatical parse operations, and identifying errors at points where sentence edits would most improve the parse quality (Prost 2009). The integration of machine learning in this domain initially involved statistical classifiers targeting specific error types, such as the method by Tetreault and Chodorow (2008) for correcting prepositional errors in English learner texts. These early methods often required distinct classifiers for each error type. Subsequently, more comprehensive models like those by Stehouwer and Zaanen (2009) employed Markov models to handle multiple confusion-prone words simultaneously. Nevertheless, these approaches were still confined to specific error types.

Modern approaches in GEC and GED treat these tasks as end-to-end machine-learning challenges, which eliminates the need to develop specific techniques for each error type. Many methods employ sequence-to-sequence architectures, similar to those used in machine translation, effectively translating from an "erroneous" version of the language to a "correct" version (Yuan and Briscoe 2016). For instance, Bell et al. (2019) used a bi-directional LSTM (as detailed in Section 4.2.5) for word-level classification, focusing on the selection of appropriate pretrained word embeddings to empower the network for this task.

Achieving human-level performance on standard English-language GEC benchmarks requires extensive training data, as demonstrated by the state-of-the-art models (Ge et al. 2018). However, this reliance on large datasets poses challenges, particularly for low-resource languages, with most research focusing on English or Chinese (Madi and Al-Khalifa 2018). The field of symbolic music error detection faces a similar issue: the quantity of available training data for any given music genre is significantly less than that for widely spoken languages.

2.2.1 Data Augmentation by Generating Errors

Data augmentation is a technique to increase the effective size of datasets for machine-learning applications. It is commonly used in GEC and GED, as data availability for these tasks is often

low and the cost of collecting new data is high. This technique usually involves introducing synthetic errors into correctly written examples of natural language, then training models to correct these errors. Synthetic errors refer to those generated through data augmentation, while natural errors are those that occur organically, such as genuine grammatical mistakes made by language learners.

Initial approaches in this area relied on basic statistics about error distributions and types to semi-randomly add errors into texts (Brockett et al. 2006; Rozovskaya and Roth 2010). Another early method involved creating detailed taxonomies of grammatical errors. For example, Foster and Andersen (2009) categorized grammatical errors into fifteen distinct types, allowing users to inject varying amounts of each error type into a corpus. Although effective, these methods typically require extensive manual coding of language rules, making them less adaptable to low-resource languages.

More recent techniques, inspired by back-translation methods in machine translation, simultaneously train models to generate and correct synthetic errors. For instance, Rei et al. (2017) streamlined the error generation process by developing models that translate entire sentences from a correct version of the language into an "error-ful" version of it. Büyük and Arslan (2021) developed an LSTM model to produce human-like misspellings in Turkish, a language with fewer resources than English or Chinese. They trained their model on a dataset of 170,000 misspelled words, enabling it to insert misspellings into a 57 million word corpus. Training a GED model on this augmented corpus achieved superior performance compared to training on a large corpus with random, non-humanlike misspellings or only the smaller dataset of misspelled words.

Htut and Tetreault (2019) evaluated a number of these models that depend on data augmentation, finding that they provide significant improvements to training on natural errors when the dataset containing natural errors is small. Notably, they found that these gains in performance from data augmentation are achievable even when the synthetic errors are qualitatively different than the natural errors; that is, a GEC method can still perform well even when it is trained on grammatical errors that a human would never make. However, when a large dataset of natural errors is available, the relative improvements obtained by augmenting further with synthetic errors are significantly diminished. Another review by White and Rozovskaya (2020) that directly compared two high-performing methods of generating synthetic errors for GEC tasks found that while neither consistently outperformed the other, their relative performance differed considerably depending on the test dataset used and the types of errors generated.

2.3 Supervised vs. Unsupervised Outlier Detection

In both real-valued sequences and natural language, a distinction should be made between *supervised* and *unsupervised* outlier (or error) detection. Supervised outlier detection is applicable when there is a large corpus of data with already labeled outliers or when data augmentation can generate additional labeled data. It operates as a conventional binary classification task. By contrast, unsupervised outlier detection is needed when the dataset contains outliers but lacks information about where they are and how many there are. For non-parametric methods like histogram- and density-based methods that require no training, this distinction alters little; there is no training step, and hyperparameters can be manually tweaked to optimize performance on a small test set without fear of overfitting to the data. However, for statistical or machine learning-based methods, this distinction influences how the task is approached.

The model-based outlier detection approach described in Section 2.1.3 involves creating a model of the primary mechanism generating a dataset, and comparing the model's behavior with observations of the dataset. In the case of unsupervised learning, where there is no clean, uncorrupted version of the mechanism's behavior as a reference, this is not possible. Aggarwal (2017, 7–8) noted that most model-based methods rely on an implicit assumption that normal behavior vastly outnumbers outliers. Under this assumption, one can train a model on data containing outliers, and treat it as if it were correct, outlier-free data, assuming that outliers are rare enough that the model will learn the data's normal behavior. However, if the data contains enough outliers or the model is sophisticated enough, the model might learn to reproduce the outliers; the model will learn to replicate both the primary and secondary mechanisms that generated the data.

Choosing the right model for unsupervised outlier detection involves a balancing act: the model needs to be sufficiently robust to understand the typical behavior of the dataset, but not so perceptive that it starts recognizing and replicating the outliers. This balancing act is a key reason for the effectiveness of autoencoders in unsupervised scenarios. Autoencoders excel at discarding costly-to-encode information, and their information retention can be finely tuned by adjusting the size of their latent space. This size is directly correlated with their reconstructive detail. However, even when models have comparable predictive capabilities, their performance can vary across different tasks, often for reasons that are difficult to analyze. Schubert et al. (2014) addressed this using the concept of *outlierness*, suggesting that different models have

inherent characteristics that make them suitable for detecting specific types of outliers. Since the definition of an outlier varies across domains, a method effective in one domain might only detect random noise in another. This ongoing research is crucial for aligning outlier detection methods across different domains.

The presence of labeled outlier examples simplifies the task, allowing for more targeted learning of relevant behaviors. Generally, incorporating any available labeled data on outliers is advantageous, even if the quantity is insufficient for training a fully supervised classifier (Aggarwal 2017, 219). Such data might be used for training a data augmentation method, as demonstrated by Büyük and Arslan (2021), or in a semi-supervised learning approach, where labeled data initiates or guides an unsupervised learning algorithm (Gao et al. 2006). The specific scenario where some outliers are labeled but the rest of the data is unlabeled is known as positive-unlabeled classification, a distinct research area (Elkan and Noto 2008). Broadly, the spectrum between supervised and unsupervised learning is fluid, with various methodologies developed for different data situations. For a comprehensive overview of these scenarios, see the summary by Aggarwal (2017, 231–235).

2.4 Methods for Taking Differences Between Sequences

A closely related concept to the idea of error detection in sequences is the concept of difference operations between sequences. Given a sequence and a corrupted version of that sequence, one could define the errors in the corrupted sequence by taking a difference between the two. Each method of taking a difference between sequences constitutes a unique definition of what the "errors" in a sequence are. In this section, I will review some common definitions of edit distance, which define distance metrics on sequences, and describe a dynamic programming algorithm for computing them. Then, I will define the closely related Needleman-Wunsch sequence alignment algorithm and its variants, which are central in defining a notion of musical error for the error detector I develop in Chapter 5.

In this section, I will use the following notational conventions and terminology:

- x and y are sequences of tokens, of length $|x|$ and $|y|$ respectively. Tokens are assumed to be categorical values rather than numeric ones.
- $x[i]$ denotes the ith token of the sequence x. $x[i : j]$ denotes the subsequence formed by ith through jth tokens from x, including the endpoints.

- The symbol ϵ denotes an empty sequence containing no tokens.

- A *prefix* of a sequence is a subsequence from the sequence's beginning up to some element, denoted $x[:i]$ for $i < |x|$. The length-0 *empty prefix* $x[:0]$ is equivalent to ϵ.

2.4.1 Edit Distances

A central concept used in many notions of sequence comparison is the notion of *edit distance*. Typically, edit distance is discussed in terms of either strings of characters or sequences of nucleotides, as its earliest applications beyond text-based string matching were in the realm of bioinformatics. For generality, I will describe these metrics as acting on sequences of generic *tokens*, which may stand in for nucleotides, characters, or musical glyphs.

Metrics of edit distance between two sequences quantify how many *operations* one would have to perform on one sequence to transform it into the other. An operation is some small, atomic change to a sequence, and each notion of edit distance defines its own set of allowable operations, with an associated *cost* for each operation. The total edit distance between two sequences is the sum of the costs of the minimal set of operations that would transform one sequence into another. The most common version of edit distance is *Levenshtein distance* (Levenshtein 1966), which takes three possible operations: deletion of a token, insertion of a token, and substitution of one token for another. Equation 2.2 defines the Levenshtein distance L between two sequences x and y recursively, in terms of the Levenshtein distance between prefixes of x and y.

$$
\begin{aligned}
L(\epsilon, \epsilon) &= 0 \\
L(x[:i], y[:j]) &= \min \begin{cases}
L(x[:i-1], y[:j]) & + C_{\text{gap}} \\
L(x[:i], y[:j-1]) & + C_{\text{gap}} \\
L(x[:i-1], y[:j-1]) & + \begin{cases} 0, & \text{if } x[i] = y[j] \\ C_{\text{sub}}, & \text{if } x[i] \neq y[j] \end{cases}
\end{cases}
\end{aligned}
\tag{2.2}
$$

Here C_{gap} is the operation cost associated with an insertion or deletion, and C_{sub} is the cost associated with a substitution. These scores can be modified depending on the domain, to encourage the edit distance to penalize certain operations more highly than others. Instead of using a single value for C_{sub}, a common practice is to use a *substitution matrix* to define a

unique substitution cost for every possible pair of tokens, penalizing the replacement of certain tokens more than others according to domain knowledge (Altschul 1991).

Other formulations of edit distance are characterized by different sets of elementary operations. The *Damerau-Levenshtein distance*, introduced by Damerau (1964), extends the Levenshtein distance by including as an operation the transposition of two adjacent elements. While the Levenshtein distance counts such transpositions as two substitutions, Damerau-Levenshtein considers it a single operation. Another variant, the *Hamming distance* (Hamming 1950), solely utilizes substitution, thereby restricting its applicability to sequences of identical length. The selection of an appropriate edit distance metric should be informed by the specific attributes of the sequences and the characteristics of the errors or variations being analyzed. For example, the Damerau-Levenshtein distance was originally formulated for analysis of spelling errors, as transposition of adjacent characters is a common mistake made by human writers. By contrast, OMR systems do not often transpose adjacent glyphs on a page, so the Damerau-Levenshtein distance would be less appropriate for analysis of OMR errors.

It is necessary to select the most suitable edit distance formulation for a given context. Since any edit distance can be adjusted to yield higher or lower values by scaling operation costs, direct comparisons of distances under different metrics are not meaningful. In situations where edit distance calculations contribute to a larger, quantifiable task, one viable approach is to experiment with various cost schemes and assess their impact on the overall system performance. Generally, the choice of an edit distance metric hinges on the nature of the sequences and the phenomena they represent. In bioinformatics, for instance, edit distances should ideally mirror biological processes like DNA mutation, so that sequences with small distances are more likely to be biologically related. This rationale underpins the use of substitution matrices based on extensive amino acid sequence analyses (Henikoff and Henikoff 1992). Conversely, for spellchecking systems, the edit distance should reflect common human typing errors, ensuring that misspelled words are closely aligned with their correct forms.

2.4.2 The Common Edit Distance Algorithm

The recurrence relation shown in Equation 2.2 is inefficient to use directly for computation of the Levenshtein edit distance, as it requires a number of function evaluations that grows exponentially with the lengths of the input sequences. There exists a canonical algorithm based on dynamic programming to efficiently compute the Levenshtein edit distance that has been

independently discovered by a large number of researchers in different fields (a survey on the topic of string matching by Navarro (2001) identifies at least six), and so goes by a variety of names. I will refer to it as the *common algorithm* for computing edit distance.

The central insight behind the common algorithm is the observation that directly evaluating the recurrence relation shown in Equation 2.3 involves repeatedly computing the Levenshtein distance between prefixes of these sequences. Instead of recomputing them on demand, it is more efficient to store the results in a table, the *score matrix* S, and retrieve them when needed. S is an $|x| \times |y|$ matrix where the entry $S_{i,j}$ is the edit distance between the prefixes $x[:i]$ and $y[:j]$. Entries outside the matrix are assumed to correspond to edit distances of ∞, while the entry at $(0, 0)$, representing an alignment between two empty sequences, is filled in with an edit distance of 0. The matrix can be filled in row-by-row by iterating the following recurrence, based on the definition of the Levenshtein distance (Equation 2.2):

$$
\begin{aligned}
S_{0,0} &= 0 \\
S_{i,j} &= \min \begin{cases}
S_{i-1,j} & + C_{\text{gap}} \\
S_{i,j-1} & + C_{\text{gap}} \\
S_{i-1,j-1} & + \begin{cases} 0, & \text{if } x[i] = y[j] \\ C_{\text{sub}}, & \text{if } x[i] \neq y[j] \end{cases}
\end{cases}
\end{aligned}
\tag{2.3}
$$

When the matrix is filled in with scores, the score at the bottom-right, at position $(|x|, |y|)$, is the edit distance between the two sequences.

Table 2.1 shows the matrix built by the common edit distance algorithm between the sequences `CDEDEAAC` and `CADEAEAC`, with a cost of 1 for the operations of deletion, insertion, and substitution. The Levenshtein distance between these two sequences, as shown by the bottom-right entry of the table, is 3.

The computational complexity of this formulation of the common algorithm is $\mathcal{O}(|x||y|)$ in both space and time. Ukkonen (1985) noted that if the edit distance between two strings is known in advance to require fewer than k insertions and deletions, then there is no need to calculate the entries of the matrix more than k positions away from the diagonal, as they represent regions of high edit distance that will not affect the final result; they will never be "chosen" by the minimum operator in the recurrence in Equation 2.3. This variant, called the

Table 2.1: The resulting score matrix for the Needleman-Wunsch algorithm when aligning the strings `CADEAEAC` and `CDEDEAAC`.

	ϵ	C	A	D	E	A	E	A	C
ϵ	0	1	2	3	4	5	6	7	8
C	1	0	1	2	3	4	5	6	7
D	2	1	1	1	2	3	4	5	6
E	3	2	2	2	1	2	3	4	5
D	4	3	3	2	2	2	3	4	5
E	5	4	4	3	2	3	2	3	4
A	6	5	4	4	3	2	3	2	3
A	7	6	5	5	4	3	3	3	3
C	8	7	6	6	5	4	4	4	3

banded edit distance, is $\mathcal{O}(k \cdot \min(|x|, |y|))$ in both space and time. If k is set at a value lower than the total number of insertions or deletions necessary to align the sequences, then the alignment will be suboptimal.

2.4.3 Needleman-Wunsch Sequence Alignment

The concept of *sequence alignment* involves positioning a set of sequences in such a way as to identify corresponding regions among them. This discussion focuses on the alignment of two sequences, a scenario that is particularly relevant to this thesis. Unlike edit distances, which provide a single numerical value indicating the difference between two sequences, sequence alignment allows for a detailed analysis of specific divergences between related sequences. This technique, originally developed for bioinformatics, is employed here to analyze differences not in genetic sequences, but in sequences representing musical content, treating discrepancies as "errors" introduced during the OMR process.

Various sequence alignment algorithms are available, many employing dynamic programming to construct comprehensive solutions from partial ones. One notable method is Dynamic Time Warping (Bellman and Kalaba 1959), which aligns sequences of continuous real-valued data. Another significant approach is the Smith-Waterman algorithm (Smith and Waterman 1981), which performs local alignments between sequences of discrete tokens, effectively identifying optimally matching subsequences. For the purposes of this thesis, however, the Needleman-Wunsch (NW) algorithm (Needleman and Wunsch 1970) is utilized. It computes a global alignment on sequences of discrete tokens and is more suited to the task at hand. Initial experiments considered the use of Dynamic Time Warping, given its prominence in Music Information Retrieval. However, its limitation to real-valued sequences rather than discrete tokens posed challenges for

representing musical symbols, leading to the selection of the NW algorithm for this research.

The NW algorithm is a method for sequence alignment, and an extension of the common algorithm used to compute edit distances. It operates on two input sequences and a defined set of operation costs. The algorithm constructs a score matrix similar to that used in computing the Levenshtein distance, as shown in Table 2.1. The key aspect of the NW algorithm is the construction of a *traceback matrix*, which is derived from the score matrix and used to find an optimal alignment path between the two sequences.

To construct the traceback matrix, the NW algorithm starts at the bottom-right cell of the score matrix and identifies which of the three options in the original recurrence relation (Equation 2.3) was used to assign a score to that cell. This involves choosing the cell with the minimum score from the cells above, to the left, and diagonally to the upper-left, with a preference for the cell to the upper-left in case of ties. This process is repeated for each cell in the matrix. Each cell in the traceback matrix then points to the cell from which it derived its score, marked by an arrow. This procedure yields an optimal path that can be traced back from the bottom-right cell to the top-left.

In practice, it is not necessary to fully construct the traceback matrix. The traceback can be computed on-demand as the algorithm progresses through the matrix, only computing entries along and adjacent to the optimal path. This optimization makes the computational complexity of computing the traceback $\mathcal{O}(\max(|n|, |m|))$, where $|n|$ and $|m|$ are the lengths of the sequences being aligned. Consequently, the total runtime of the NW algorithm is primarily determined by the time taken to create the score matrix.

The alignment can be inferred from reading the path once more, this time in reverse, from top-left to bottom-right. Wherever the path passes down diagonally, that corresponds to a match or a mismatch between tokens. Where the path passes horizontally, that corresponds to an insertion operation; where it passes vertically, that corresponds to a deletion. It is possible that multiple valid paths exist that correspond to minimal sets of operations with equal cost; this corresponds to situations where there are score ties when building the traceback matrix. Tiebreaking by always preferring to extend the path diagonally ensures that, of all valid alignments, the algorithm finds the alignment that requires the fewest insertions and deletions, and with the most substitutions.

Table 2.2: Traceback matrix for the Needleman-Wunsch algorithm. Each cell contains an arrow pointing to the minimal value of its upper, left, and upper-left neighbors. The shaded path is the one identified by the NW algorithm as corresponding to a minimal set of operations. The symbol ϵ denotes an empty prefix: the first zero characters of each string.

	ϵ	C	A	D	E	A	E	A	C	
ϵ	○	←	←	←	←	←	←	←	←	
C	↑	↖	←	←	←	←	←	←	←	
D	↑	↑	↖	↖	←	←	←	←	←	
E	↑	↑	↖	↖	↖	←	←	←	←	C-DEDEAAC
D	↑	↑	↖	↖	↑	↖	←	←	←	CADEAE-AC
E	↑	↑	↖	↑	↖	↖	↖	←	←	
A	↑	↑	↖	↑	↑	↖	←	↖	←	
A	↑	↑	↑	↑	↑	↑	↖	↑	↖	
C	↑	↑	↑	↖	↑	↑	↑	↑	↖	

2.4.4 Affine Needleman-Wunsch Sequence Alignment

The Affine Needleman-Wunsch (ANW) algorithm modifies the traditional Needleman-Wunsch approach to accommodate scenarios where long contiguous gaps (representing deletions or insertions) in a sequence alignment are less significant than multiple isolated gaps. This modification is particularly relevant in fields like OMR where a physical imperfection on a page, like a smear of ink, might obscure several consecutive glyphs, causing them to be missed by the OMR algorithm. In such cases, it is beneficial to adjust the alignment algorithm to reflect these long gaps more accurately, mirroring the nature of the errors. The ANW algorithm implements *affine gap penalties* to achieve this. Instead of a uniform cost for gap operations (deletions and insertions), the algorithm assigns a higher cost to initiating a gap and a lower cost to extending an existing gap. This approach encourages the algorithm to prefer alignments with fewer, longer gaps, which can more accurately represent the errors caused by processes like smudging in OMR.

Developed by Altschul (1986), the ANW algorithm uses additional matrices to track the costs of gaps in each sequence. Besides the standard score matrix S, it employs a gap-in-X matrix L^x and a gap-in-Y matrix L^y. The recurrence relations used to populate these matrices, as detailed in Equation 2.4, distinguish between the cost of opening a gap, denoted as C_{open}, and the cost of extending a gap, denoted as C_{gap}. These separate costs allow the algorithm to differentiate between starting a new gap and continuing an existing one, effectively applying affine gap penalties.

$$
\begin{aligned}
S_{i,j} &= \min\left(S_{i-1,j-1},\ L^x_{i-1,j-1},\ L^y_{i-1,j-1}\right) + \begin{cases} 0, & \text{if } x[i] = y[j] \\ C_{\text{sub}}, & \text{if } x[i] \neq y[j] \end{cases} \\
L^x_{i,j} &= \min\left(S_{i-1,j} + C_{\text{open}} + C_{\text{gap}},\ L^x_{i-1,j} + C_{\text{gap}}\right) \\
L^y_{i,j} &= \min\left(S_{i,j-1} + C_{\text{open}} + C_{\text{gap}},\ L^y_{i,j-1} + C_{\text{gap}}\right)
\end{aligned}
\tag{2.4}
$$

Here each matrix keeps track of each type of operation. Using these recurrences, the three matrices can be filled from top-left to bottom-right, using their own and each others' entries.

The process of constructing the traceback matrix is similar to the non-affine case, except that for each step in the path's development, the algorithm chooses the cell with the minimum possible value across all three matrices: S, L^x, and L^y. Ties are again broken by prioritizing the path on the diagonal. After the traceback matrix is constructed, the optimal path through the matrix is found using the same method as the non-affine case.

`A-A-AC`	`AA--AC`
`ABABAC`	`ABABAC`
$C_{\text{open}} = 0, C_{\text{gap}} = 1, C_{\text{sub}} = 2$	$C_{\text{open}} = 2, C_{\text{gap}} = 1, C_{\text{sub}} = 2$

Table 2.3: Two sequences and their alignment under different affine gap penalty cost schemes. C_{open} Is the cost of opening a gap, C_{gap} is the cost of extending a gap that is already open, and C_{sub} is the cost of replacing one token with another.

Table 2.3 illustrates the impact of using affine gap penalties in aligning two token sequences, `AAAC` and `ABABAC`. The example demonstrates how the ANW alignment, with appropriate gap penalties, can lead to different optimal alignments compared to the regular NW algorithm. In the example provided, with $C_{\text{open}} = 0$, the cost scheme mirrors the unmodified NW alignment. However, the alignment using affine gap penalties (on the right) shows an additional mismatch to merge two short gaps into a larger one, which is a characteristic feature of the ANW approach. This highlights the flexibility of the ANW algorithm in handling sequences where long gaps are more representative of the underlying phenomena than multiple isolated gaps.

The ANW algorithm is used extensively in the OMR error detection process I develop in Chapter 5. I have implemented an algorithm for computing the ANW alignment between two sequences in the Python programming language as a standalone package. The method can align sequences containing arbitrary categorical data, and can operate with a specified substitution matrix. I also implement the banding technique by Ukkonen (1985) to reduce the total time

complexity of building the score matrices to $\mathcal{O}(k \cdot \min(|x|, |y|))$. Python is an interpreted language, and so it is slow to perform the kinds of repetitive arithmetic operations necessary to build the score matrices. Since the construction of the score matrices dominates the running time of the algorithm, I implement this step using the Numba just-in-time compiler for Python (Lam et al. 2015), which compiles the matrix-building step directly to machine code, making the overall single-threaded performance of my implementation comparable with one written directly in a low-level language.

2.5 Error Detection and Correction in Symbolic Music

Here, I review literature on in the domain of symbolic music that addresses the detection and correction of errors in musical content. This review will cover literature that directly addresses these tasks or is related and could potentially be adapted for error detection and correction in symbolic music.

The scope of this review excludes the domain of audio-based error detection, as this area typically does not address errors in musical content, which is the primary concern of the error detector developed in Chapter 5. Audio-based methods often involve statistical techniques for noise reduction or removal in audio recordings, applicable to both musical and non-musical contexts. These approaches align with the real-valued sequence error detection methods discussed in Section 2.1.3. For a comprehensive review into statistical techniques in audio outlier detection, see Godsill et al. (2001). Autoencoders are a prevalent tool in audio noise reduction, exploiting the characteristic of noise being generally incompressible (Abouzid et al. 2019). In the context of musical audio error detection, some studies do consider musical content, but typically through a preprocessing step that transforms the audio into a symbolic format or a simplified representation that approximates notes and durations. For example, Bhargave (2019) developed a system that runs a pitch estimation system on recordings of improvisations in Hindustani classical music and identifies places where the musician has played a note that is incompatible with the current raga (roughly, the equivalent of the Western "scale").

Thus, the focus of this review remains on symbolic music, examining methodologies and technologies that detect and correct errors directly within symbolic musical representations.

2.5.1 Error Analysis in Music Transcription Tasks

Many OMR systems incorporate components for musical error checking either as a post-processing step or as an integral part of the OMR process to assist in symbol identification. Early OMR methodologies primarily focused on metrical accuracy, ensuring that the correct number of beats would be present in per measure in the output (Kato and Inokuchi 1992; Coüasnon and Retif 1995). An approach by Droettboom et al. (2002) incorporated a metric correction post-processing phase involving checks for common OMR errors that could lead to incorrect interpretation of musical durations, such as mistaking specks of dust for augmentation dots.

Rossant and Bloch (2006) assessed OMR outputs against pre-defined rules of musical syntax concerning meter, accidentals, and tonality. This process combined with confidence assessments from earlier OMR stages to identify potentially incorrect detections. In mensurally notated music, Thomae et al. (2022) automated the correction of OMR outputs by identifying notes that violated traditional counterpoint rules, treating these violations as potential OMR errors. Their study included a user trial where an expert transcriber corrected OMR outputs with and without the aid of the automatic counterpoint error detector. The study found a nearly 50% reduction in correction time when the error detector was used, providing some validation for similar approaches. Although my research does not include such a user study, the success of the method by Thomae et al. suggests that a reliable error detection system could significantly reduce the time required for manual corrections in OMR processes.

Error correction in music is often considered as a step in Automatic Music Transcription (AMT) methods. Cogliati and Duan (2017) designed a sequence alignment method to align AMT outputs with ground truth, tuning their method with human assessments of transcription quality. Another notable contribution is by McLeod et al. (2020), who developed the Musical Instrument Digital Interface (MIDI) Degradation Toolkit, a software tool designed to introduce errors into symbolic music in a controlled and customizable manner. This toolkit aids in analyzing AMT output errors and provides rule-based and machine-learning models for error detection and correction, although they primarily serve as foundational examples for defining these tasks.

Symbolic music transduction is another relevant task, focusing on converting raw note probability outputs from AMT models into valid symbolic music files. Common methods include global thresholding of note probabilities or using hidden Markov models to filter these probabilities. Advanced techniques like the recurrent neural network approach by Boulanger-Lewandowski et al. (2013) have shown substantial improvements over traditional methods. Furthermore, Ycart

et al. (2019) compared various machine-learning models for symbolic music transduction, suggesting that incorporating a language model-based transduction as a post-processing step may not offer significant benefits over training AMT systems for direct binarized symbolic music output.

Outside of AMT and OMR, applications of error detection in symbolic music are limited. Sidorov et al. (2014) explored the use of formal grammars in music analysis, hypothesizing that correct music is highly compressible and deviations significantly increasing the information content in a piece are likely errors. Additionally, Nakamura et al. (2017) used error detection as part of a method of aligning two musical signals where one signal is considered ground-truth and the other is a human performance that may contain errors (where an error is defined as any deviation from a given musical score).

2.5.2 Notions of Musical Edit Distance

The application of edit distance measures in Music Information Retrieval (MIR) and their relevance to the concept of musical error have been explored in various studies. Habrard et al. (2008) utilized the Levenshtein distance to compare monophonic melodies, classifying melodies as variations of one another based on low distance scores. Lemström and Pienimäki (2007) expanded the discussion on the utility of edit distances in musical retrieval tasks.

The evaluation of Optical Character Recognition (OCR) outputs, closely related to OMR, often employs edit distance measures. This connection has motivated research into tailoring edit distances specifically for OMR evaluation. Bellini et al. (2007) suggested analyzing OMR errors in terms of the time required for a human to correct them, introducing a weighting system based on the estimated correction time for each symbol in Western music notation. These weights, combined with edit distance concepts, aim to offer a meaningful and interpretable measure of OMR performance that is discussed further in Section 3.2.2.

However, the practicality and universality of such weighted edit distances are questioned. Byrd and Simonsen (2015) argued that accurately quantifying correction time is challenging due to the variety of user interfaces available for musical score correction. They emphasized the complexity and diversity of Western music notation, doubting the feasibility of a universally applicable metric that accurately weighs the importance of individual musical glyphs across all music genres. Supporting this skepticism, Hajic et al. (2016) examined several OMR evaluation metrics, including variants of the Levenshtein distance, and observed significant discrepancies in

how these metrics align with each other. This research underlines the profound impact of error definition on both the identification of erroneous sections in OMR output and the measured accuracy of OMR systems.

MusicDiff (Foscarin et al. 2019) is a tool designed for computing differences between two MusicXML format score files. Unlike other methods that focus solely on audible differences, MusicDiff identifies visible notation differences, such as differentiating between beamed and unbeamed eighth notes. This feature makes it particularly suitable for evaluating OMR outputs. MusicDiff represents music as a hierarchical tree structure and detects errors through insertions, deletions, and substitutions within this tree, utilizing a version of the Levenshtein distance metric. The tree-based representation is adept at handling complex beaming configurations and nested tuplets, which pose challenges for purely sequence-based symbolic music representations. However, this approach has limitations; if an input symbolic music file contains musically implausible content (e.g., ties between non-consecutive notes or impossible beaming configurations), the file cannot be transformed into a well-formed tree structure, and thus MusicDiff cannot process it. Such errors are not uncommon in OMR outputs.

Mupix, developed by Daigle (2020), offers a different approach for comparing MusicXML files, particularly aimed at evaluating OMR results. Unlike MusicDiff, Mupix does not convert music into a tree structure. Instead, it parses each file into seven distinct arrays, each representing a different type of musical object (notes, rests, clefs, keys, dynamics, time signatures, and spanners). It applies the NW algorithm (discussed in Section 2.4.3) to align these arrays, facilitating the comparison of corresponding elements across files. Each category of musical objects allows for direct comparison within its group (e.g., notes with notes), and Mupix calculates a similarity score for each array and an overall score for the two input files. This method's flexibility allows it to handle malformed or corrupted MusicXML files. However, due to its custom similarity functions and specific categorization scheme, Mupix would require modifications to effectively work with non-Western music notations or any variants of Western music notation with symbols outside its tested Common Western Music Notation (CWMN) corpus.

Chapter 3

Optical Music Recognition

In this chapter, I explore Optical Music Recognition (OMR) to contextualize my work in error detection for Optical Music Recognition (OMR) output. While my work does not concern the development of new OMR methods, its intended application is to correct the output of OMR systems, and so its design will be informed by the nature of OMR systems so as to make it maximally useful to the human corrector. I concentrate on features of OMR systems pertinent to error detection, specifically focusing on printed scores in Common Western Music Notation (CWMN), as my method is designed for this context. Older notations and handwritten notation techniques will be mentioned when relevant for historical or technical understanding.

Section 3.1 begins by outlining the objectives and application domains of Optical Music Recognition, followed by a classification of various OMR tasks. I examine the complexities that make Common Western Music Notation (CWMN) challenging to interpret compared to regular text. To provide context, I define CWMN with a short history of Western musical notation, highlighting the range of challenges that OMR systems need to address in order to handle the diverse notational phenomena present in CWMN. I detail the standard sequence of operations commonly used in OMR and discuss recent machine-learning approaches that aim to tackle OMR challenges in a more integrated manner.

Evaluating OMR outcomes and contrasting various OMR systems is difficult due to the complex, hierarchical characteristics of music notation and the disparate objectives among different OMR systems. In Section 3.2 I delineate the challenges in OMR evaluation along three axes: the choice between assessing performance at the symbol level or the semantically encoded output level, the method of weighing different types of errors, and the consideration of whether to evaluate OMR systems independently or within the scope of broader music information retrieval

processes.

Finally, in Section 3.3, I describe the commercial OMR software PhotoScore 8.0.4, which serves as the data source for OMR errors that I use in training the error detector elaborated in Chapter 5. I provide details on the software's functionalities, its in-built error correction and detection mechanisms, and examples of the common errors it tends to generate when processing scans of various qualities.

3.1 An Overview of Optical Music Recognition

OMR is a field of study that concentrates on the problem of translating scanned images of musical notation into symbolic, machine-readable forms. It encompasses many applications and sub-tasks, each with its own definition of what goes into an OMR system and what comes out; to encompass this variety, Calvo-Zaragoza et al. (2021) propose the definition:

> Optical Music Recognition is a field of research that investigates how to computationally read music notation in documents.

A primary motivation for the field of OMR is that of archival and preservation, and improving access to cultural heritage that may otherwise be lost or forgotten. Most music written before the digital age exists only in the form of physical handwritten or typeset musical scores. While technological advancements have led to an increase in the volume of musical material that is being scanned and made publicly available by digital means, most of these scanned works exist only as image files, their musical contents still not machine-readable (Pugin 2015). Transcribing musical scores into digitally accessible forms by hand can be a tedious, difficult process, and notation editing software often demands a steep learning curve, requiring transcribers to develop proficiency through repeated use (See the "World Cup of OMR" described by Daigle (2020) for an evaluation of how fast expert transcribers are able to copy polyphonic scores from image files into notation editors). As a result, transcription of large music archives by hand is an impractical goal. The promise of OMR is the possibility of mass automatic or near-automatic conversion from scanned images to machine-readable symbolic music files in a variety of formats, enabling new types of musicological study and new options for performers to choose from. In particular, OMR has the potential to be used in low-resource situations to preserve objects of cultural heritage in countries where the funding would not otherwise exist to transcribe them (Thomae 2023).

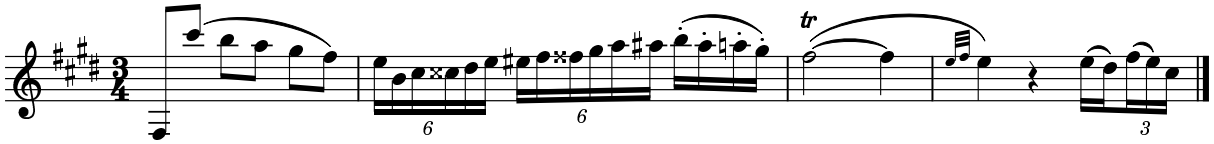
As Calvo-Zaragoza et al. (2021) have noted, while OMR is sometimes referred to as a singular "task" or "process," it encompasses a diverse range of possible tasks. Their proposed taxonomy for OMR systems covers several axes along which these tasks can differ:

- Is the music typeset or handwritten?
- Is the music in CWMN, an earlier form of Western notation, non-Western music notation, tablature, or something else?
- What is the quality of the document being fed into the OMR system? Is it a direct export from a notation program, having never been printed on paper and scanned? Is it a high-quality scan, or is it heavily degraded to the point of being in parts illegible?
- How complex is the score, structurally?
- What is the output format? MusicXML, MIDI, a project file for a particular notation editor, or is it to be rendered directly to audio?

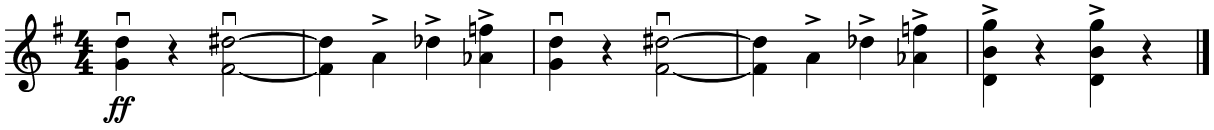
On the notion of score complexity, Byrd and Simonsen (2015) define four categories useful for describing the capabilities of OMR systems:

- *Monophonic* scores contain a sequence of non-overlapping notes on each staff.
- *Homophonic* scores may contain chords, but only ever as a single voice per staff.
- *Polyphonic* scores may contain multiple voices on one staff, each of which can play multiple notes at once and act rhythmically independent to the other voices. It is always possible to transcribe a staff of polyphonic music into multiple homophonic staves by separating out the voices.
- *Pianoform* scores contain interaction between different staves; note the beam that crosses between staves in Figure 3.1d. They cannot always be separated into individual voices while still retaining all the information from the original score.

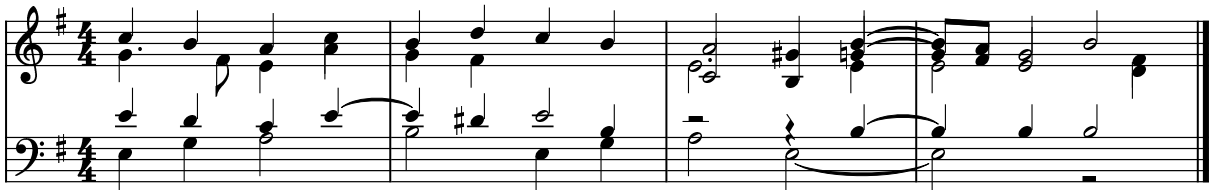
Figure 3.1 contains examples of each of these types of notation. The terms "homophonic" and "polyphonic" here differ in meaning from the notions of homophony and polyphony as they appear in music-theoretical texts. The term *homophony* in music theory is a textural descriptor referring to a melody that is harmonized by rhythmically similar, but not identical parts. The



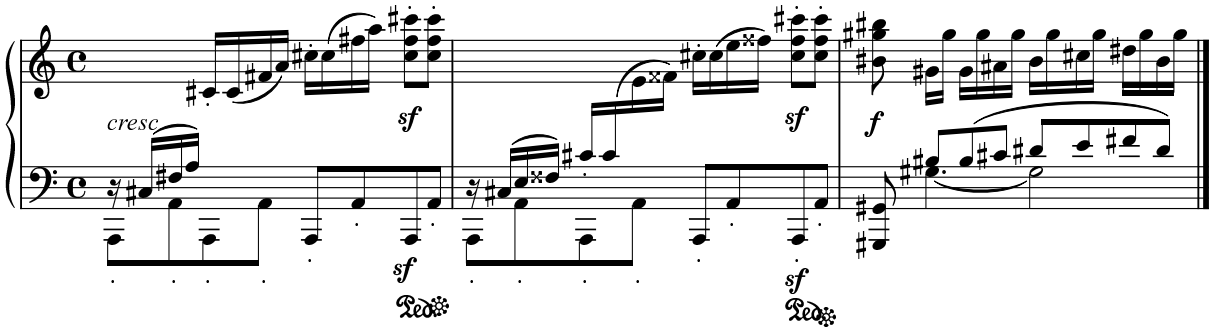
(a) An example of a Monophonic score: Wolfgang Mozart, Concerto in A Major K.622, mm. 448–411, clarinet part.



(b) An example of a Homophonic score: Claude Debussy, String Quartet No. 10, Mvt. 4, mm. 252–256, second violin part.



(c) An example of a Polyphonic score, a four-part choral work engraved on two staves: Thomas Tallis, *Lamentations of Jeremiah the Prophet*, mm. 10–14.



(d) An example of a Pianoform score: Ludwig van Beethoven, Piano Sonata No. 14, Mvt. 3, mm. 7–9.

Figure 3.1: Examples of the categories of score complexity for the purposes of categorizing OMR systems.

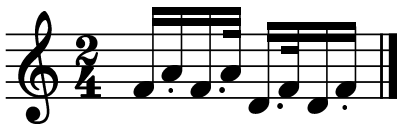


Figure 3.2: A brief snippet of music showing how symbols can be confused for one another based on their proximity to other symbols.

example in Figure 3.1c is texturally homophonic, but for the purposes of categorization for OMR, this particular engraving is polyphonic. The above definition of "Homophonic" is more similar to the textural descriptor *homorhythmic*, which refers to a homophonic score containing voices that all play in the same rhythm; however, I will continue to use the terms as defined above for the sake of concordance with other OMR literature.

3.1.1 The Difficulties of Reading Common Western Music Notation

The field of OMR is sometimes compared to the field of Optical Character Recognition (OCR) by analogy; some literature from early in the field's history refers to it explicitly as "optical character recognition for music" (Kassler 1972). This metaphor is not wholly incorrect, but CWMN is significantly more complicated than the written word. The most significant difference between the fields of OCR and OMR is that text is inherently one-dimensional and CWMN is not. For most natural languages, there is no semantic meaning to a glyph being higher or lower on the page than its neighbors, and within a single line of text the order in which characters should be read is unambiguous. In polyphonic CWMN, multiple notes occur at the same time, notes may begin or end while other notes are still ringing, and the reading order of symbols within a single staff may not be uniformly left-to-right (for example, when a cluster chord forces some notes that are to be played simultaneously to the left or right, for the sake of fitting them all on a single stem). Even in monophonic CWMN multiple symbols (articulation markings, dynamics, tuplet indicators) can occur vertically stacked at the same horizontal location on the same staff. For printed text, handwritten text, and historical documents, in OCR characters are generally segmented from each other based on their horizontal positions within text strips (Likforman-Sulem et al. 2007), but in OMR symbols must always be interpreted in both the horizontal and vertical dimension.

The other significant difference between the fields of OCR and OMR is that the semantic meaning of each symbol on a page of CWMN is heavily context-dependent. Many symbols only carry meaning when associated with adjacent notes, like ties, slurs, dots of augmentation and

articulation markings, and these must be correctly associated with neighboring symbols on the page. This can be a challenge in dense polyphonic music (See Figure 3.1d), where such markings may be in the proximity of several note heads at once. For an example of musical notation that can cause problems for OMR systems in this way, see Figure 3.2. Which dots in this measure are dots of augmentation and which are staccato markings? It is obvious to a musician given this clean, digitally engraved image: the staccato markings are smaller than the augmentation dots and perfectly centered underneath their noteheads, and the thirty-second notes necessitate longer notes preceding them. Even a small smudge or error in binarization could easily cause an OMR system to miss these cues and so misinterpret some of these symbols. This is an extreme example, but such densely engraved, articulation-heavy passages occur frequently in, for example, some editions of Mendelssohn's string quartets (deGroot-Maggetti et al. 2020).

Worse, correctly interpreting a symbol may rely on interpretation of other symbols many measures away, even in monophonic music. When key signature changes interact with natural signs, accidentals, courtesy accidentals, ties, and clef changes, the number of factors involved in interpreting the pitch and duration of a single note can become complex. Note the tie and slur that begin on the F♯ half-note in the third measure of Figure 3.1a; in order to determine which one is a tie and which one is a slur, it is necessary to correctly parse the pitches of the two following quarter notes, and to determine which one lies at the end of which arc. See Byrd and Simonsen (2015) and Calvo-Zaragoza et al. (2021) for further descriptions of this complex context-dependent behavior and examples of phenomena in musical scores that OMR processes have difficulty with.

Further complicating OMR is the fact that CWMN is not a standardized or well-defined system. CWMN has its origin in *neumes*, a method for notating melodies as annotations above written text. Neumes first operated as mnemonic aids, noting only melodic gestures to aid the performer in remembering a melody that was transmitted orally. By the eleventh century, neumes evolved to be placed at varying heights above lyrical text to denote absolute pitch, and then with reference to a single staff line scratched across the parchment of the page. Over time, more lines were added, and a convention was introduced of writing an uppercase "C" or "F" on one of the lines to mark the pitch of that line: the antecedent to the modern clef (Strayer 2013). Later developments, in the twelfth century onwards, introduced rhythmic notation to distinguish between multiple duration values of notes. Originally, these multiple duration values did not fully describe the duration of a note as it should be performed, and the intended duration

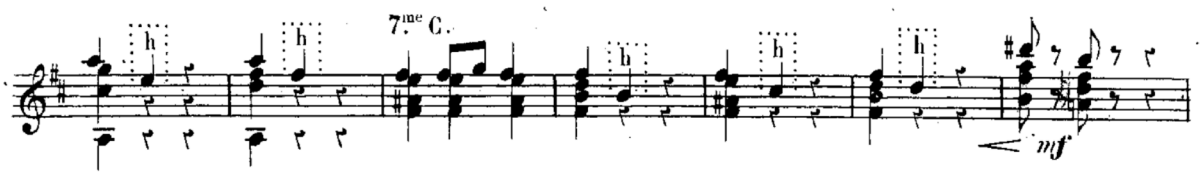
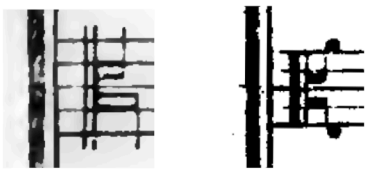


Figure 3.3: Napoleon Costé's Adagio et Divertissement pour la Guitare, op. 50, Mvt. 2, mm. 23–29, featuring a now-obsolete typesetting of the quarter rest that resembles a mirrored version of the modern eighth rest.



(a) (b)

Figure 3.4: Two variants of the alto clef in use until the twentieth century.

of each note could only be inferred from its musical context. This was the case in Mensural notation (Thomae 2023) and Modal notation (Rastall 1982, 38–39). In the seventeenth century this behavior was simplified with the concept of *orthochronic* notation, in which the notated duration of each event on the page always reflects its exact duration in performance (ibid., 97–98). By the end of the seventeenth century, round noteheads became more common than square- or diamond-shaped ones, the use of the treble clef for upper voices was standardized, and both vocal and instrumental notation used the same five-line staff. Rastall (1982, 175–177) identifies this as the point from which modern audiences would, for the most part, have little problem interpreting and performing music printed in this era.

For the purposes of OMR, the end of the seventeenth century marks the beginning of what is referred to today as Common Western Music Notation (Calvo-Zaragoza et al. 2021). However, the conventions of CWMN have not remained totally static in the three centuries since then. Symbols like the double-flat 𝄫 and double-sharp 𝄪 were not introduced until partway through the eighteenth century, and the use of the natural sign ♮ to cancel accidentals was not standard throughout Europe until the end of the eighteenth century (Rastall 1982, 182). Several alternate methods of writing the alto clef persisted during this period, shown in Figure 3.4. Well into the twentieth century, some French publishers used a horizontally mirrored version of the modern eighth rest as a quarter rest, shown in Figure 3.3.

While some rules had become standardized and many older symbols and clefs had become obsolete by the beginning of the twentieth century, Western twentieth-century composition saw

an explosion in the use of new techniques, instrument-specific articulation methods, and compositional styles too metrically or tonally complex to be written down using the conventions of CWMN up to that point. Some of these new notation types have become standard parts of CWMN over time and are included in common notation editors, but many have multiple competing implementations and remain unstandardized. See Stone (1980) for a review of the standards of CWMN as they existed at the turn of the twentieth century, as well as a catalogue of extensions developed in the decades following.

In summary, OMR is a more difficult task than OCR for a variety of reasons: 1) the layout of musical symbols on a staff is more complex than the layout of written text, 2) the meaning of every musical symbol is heavily context-dependent, so reading music requires a step of interpretation that is not present in reading text and 3) the rules by which musical symbols are interpreted themselves change over time to adapt to the needs of performers and composers. As a result, OMR algorithms are prone to errors, and the context-dependent nature of musical notation means that errors in the OMR process beget further errors as symbols are interpreted in the context of other incorrectly interpreted symbols.

3.1.2 The Traditional OMR Pipeline

The field of OMR has a long history, with a wide variety of approaches implemented in both academic research and commercial products. Here I do not aim to comprehensively review all existing methods for OMR, but to describe at a high level all the techniques and pipelines for OMR that have been proposed, focusing primarily on those relevant for the case of printed CWMN. I also will omit details on the act of scanning physical documents, dealing with cameras, lenses, image file types, and image resolution, though some papers treat this as an integral part of the OMR pipeline. A number of reviews have been conducted on the field of OMR that give a more comprehensive overview of the technical and practical aspects of creating an OMR system. See Novotný and Pokorný (2015), Shatri and Fazekas (2020), and Calvo-Zaragoza et al. (2021) for thorough theoretical overviews of the field, Rebelo et al. (2012) for an overview focusing on handwritten music recognition, and Helsen et al. (2014) for a description of the challenges of OMR on historical manuscripts.

OMR was first attempted in the 1960s, in a system that only handled a small subset of possible symbols in homophonic CWMN, focusing primarily on recognizing note heads and distinguishing between different durations (Pruslin 1966). It accomplished this by binarizing

score images, using simple edge detectors on these binarized images to emphasize vertical and horizontal lines, identifying individual staves, and then removing staff lines and beams. The final step of the process was an analysis of the connected components[1] that remained on the page, which were all assumed to be notes, beams, or flags of eighth notes. Prerau (1975) extended this work to cover a larger subset of CWMN symbols, including accidentals, time signatures, key signatures, clefs, and bar lines. To do this, Prerau made a complex series of hand-coded rules to identify connected components on a page of music by their dimensions and their relative positions to the staff and to identified notes; for example, flats were differentiated from natural signs by noting that connected components containing natural signs tend to be slightly taller. While successful, these rules were only valid for the particular musical font and engraving system that was analyzed to create them.

The pipeline of image processing, staff-line identification, staff-line removal, and connected component analyses established by Pruslin and Prerau was expanded upon in future approaches. Most reviews of OMR literature describe OMR in terms of *subtasks* running in series. For example, in a review of OMR systems up through the early nineties, Blostein and Baird (1992) found that most follow a similar step-by-step procedure, which they describe as a three-step process: symbol recognition, 2D symbol arrangement, and high-level recognition. The number and nature of these subtasks varies between authors, with subtasks increasingly added or broken into heirarchies of smaller tasks as the field develops. Here I define a list of OMR subtasks, based off of a description of typical OMR architectures from Rebelo et al. (2012).

1. **Image preprocessing:** Cleaning up an image in preparation for the OMR process. This step often involves image binarization, (turning a grayscale or color image into black-and-white), itself a difficult task (Mustafa and Kader 2018). Specialized techniques have been developed to deal with binarization of highly degraded historical documents (Sulaiman et al. 2019). Different OMR systems assume different quality of scans as input, so this step can involve only binarization or a host of operations like de-skewing to remove the effect of curled pages and noise removal. Where scans are exceptionally low-quality or degraded, this may include performing morphological operations on images to fill holes in lines or note heads, making the succeeding OMR steps more reliable (Goecke 2006).

2. **Staff processing:** Recognition of staff lines. This comprises several tasks that are some-

1. In the context of image processing on binarized images, a connected component is a contiguous group of black pixels in an image that is surrounded by white pixels.

times treated separately: location of staves on a page, grouping staves into systems, marking the locations of staff lines, and sometimes removing staff lines from the image. A variety of methods exist for staff line identification, including simple approaches that assume staves to be near-perfectly horizontal on the page (Pruslin 1966), methods of projection (Fujinaga 1988), and template matching on thin vertical slices of the page (Bellini et al. 2001). Curvature of staff lines due to the curve of photographed book pages is a difficult problem in this area; dynamic programming-based methods have been developed that can adapt to slight curvature (Miyao and Okamoto 2004).

3. **Music symbol isolation:** Segmentation of the page into individual symbols. This may be accomplished using connected component analysis if the staves have been removed from the image. For some types of monophonic or homophonic music, segmentation may occur only in one dimension, wherein a staff is split at many horizontal points along its length such that each remaining slice contains only a single musical symbol, as in Bellini et al. (2001). Often this step involves further decomposing symbols on the page into *primitive shapes* (Bainbridge and Bell 2001). A primitive shape is a graphically meaningful section of a musical symbol that is itself indivisible into any other shapes; for example, a barred eighth note is composed of the primitive shapes of a note head, a beam, a stem, an accidental, and so on. Different approaches give different definitions for how to break down the musical surface into primitive shapes, but the impetus behind using them is that they allow breaking down intricate, complicated connected components on a page into smaller, simpler fragments that are easier to reliably categorize and detect.

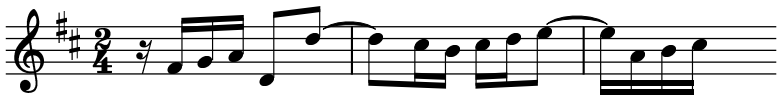
4. **Music symbol classification:** Classification of each of the symbols or primitive shapes found in the previous step as musical objects. This step can contain nearly any image-classification method. Prerau (1975) primarily used the size of the tighest bounding box around each segmented symbol as a feature to classify them by, while Fujinaga (1999) used k-nearest-neighbors on a small set of extracted features from each symbol. Bellini et al. (2007) used a feed-forward neural network operating directly on the pixel values of each symbol. The output of this process is also called an *agnostic encoding* of a musical score, as it is agnostic to musical semantics. See Figure 3.5b for an example of an agnostic encoding of a musical excerpt.

5. **Music notation reconstruction:** Recombination of primitive shapes back into musically

meaningful symbols based on their relative proximity, and analysis of each symbol's pitch and duration. For example, this step involves the combination of a flag, beam, stem, accidental, notehead, dot of augmentation, and articulation marking into a single eighth note with a particular pitch. This step is also referred to as translation from an agnostic encoding to a *semantic encoding* of a score. See Figure 3.5c for an example of what such a semantic encoding might look like. This is often done by encoding rules of musical notation into grammars, in an analogous fashion to how an OCR system may encode rules of syntax to ensure that its output is well-formed. As an example, the grammar implemented by Reed and Parker (1996) defined how primitive shapes must be arranged with respect to one another, and it acts as a kind of template that the identified shapes can slot into. However, errors in previous steps of the OMR pipeline can lead to situations that are grammatically impossible (e.g., a beam or flag without any accompanying note, a tie that begins but has no end note), so these grammars must be flexible enough to account for any possible input. See Section 2.5.1 for further examples of applying musical rules within the OMR process to detect errors.

6. **Encoding of music notation into output format:** Translation from the method's internal representation of labeled glyphs into a symbolic format such as MusicXML, Musical Instrument Digital Interface (MIDI), Music Encoding Initiative (MEI). Calvo-Zaragoza et al. (2021) break down this step further, distinguishing between systems that output a playable version of the original score (usually as MIDI) but do not attempt to retain all of the score's information, and those that attempt to faithfully encode all the information in the original score (usually as MusicXML). They named the latter case *structured encoding*, and noted that this is the most common end-point for commercial applications, while few OMR systems originating in academia reach this level.

While not typically categorized as a subtask within the OMR pipeline, significant research has been devoted to post-processing steps. These steps involve evaluating the recognized and encoded music against the norms of musical genres or rules of notation to identify or automatically rectify errors in OMR output. The error detector detailed in Chapter 5 is one such post-processor. These approaches can be broadly classified into two categories: those based on statistical or machine-learning models of symbolic music, and rule-based systems. For instance, Church and Cuthbert (2014) utilized analyses of rhythmic repetition within musical pieces to



(a) Music excerpt.

```
clef.G-L2, accidental.  sharp-L5, accidental.sharp-S3, digit.2-L4,
digit.4-L2, rest.sixteenth-L3, note.beamedRight2-S1, note.beamedBoth2-L2,
note.beamedLeft2-S2, note.beamedRight1-S0, note.beamedLeft1-L4,
slur.start-L4, barline-L1, slur.end-L4, note.beamedRight1-L4,
note.beamedBoth2-S3, note.beamedLeft2-L3, note.beamedRight2-S3,
note.beamedBoth2-L4, note.beamedLeft1-S4, slur.start-S4, barline-L1,
slur.end-S4, note.beamedRight2-S4, note.beamedBoth2-S2, note.beamedBoth2-L3,
note.beamedLeft2-S3
```

(b) Agnostic encoding of the music excerpt.

```
clef-G2, keySignature-DM, timeSignature-2/4, rest-sixteenth, note-F#4_sixteenth,
note-G4_sixteenth, note-A4_sixteenth, note-D4_eighth, note-D5_eighth,
tie, barline, note-D5_eighth, note-C#5_sixteenth, note-B4_sixteenth,
note-C#5_sixteenth, note-D5_sixteenth, note-E5_eighth, tie, barline,
note-E5_sixteenth, note-A4_sixteenth, note-B4_sixteenth, note-C#5_sixteenth
```

(c) Semantic encoding of the music excerpt.

Figure 3.5: An example of a possible agnostic and semantic encoding of a musical excerpt. Reproduced from Figure 1 of (Thomae et al. 2020).

detect and correct metrical errors in symbolic music output from OMR processes. Thomae et al. (2022) developed a system encoding the rules of counterpoint in Renaissance polyphony to pinpoint errors in OMR outputs and flag them for human review. Similarly, Jin and Raphael (2012) applied dynamic programming techniques to identify sequences of symbols that agree maximally with OMR outputs while adhering to the constraints of the piece's time signature. Approaches like those proposed by Sands (2020) and Zhang (2017) focus on validating OMR outputs against MIDI files containing corresponding musical information, employing sequence alignment methods similar to those discussed in Section 2.4.3 to compare the two representations.

Table 3.1 shows a list of possible failure modes for each OMR subtask, and examples of how these failure modes can manifest as errors in the OMR output. Errors in tasks early in the pipeline tend to affect the performance of tasks later in the pipeline; noise introduced by the image preprocessing step can be misidentified as musical symbols in the symbol classification step, which may then cause issues in reconstructing notation. Bellini et al. (2007) labeled errors caused by poor performance in previous steps *secondary errors*, and called errors made solely within a step *primary errors*.

A solution to the problem of cascading secondary errors is to perform multiple sub-tasks at

Table 3.1: Possible failure modes for each step in the traditional OMR pipeline, and examples of possible effects that each failure can have on the output of the system.

OMR Subtask	Failure mode	Possible effect on output
Image preprocessing	Binarization failure	No staves found
	Binarization blanks out regions of page	Regions of page missing from output
	Insufficient denoising	Symbols incorrectly recognized, noise interpreted as symbols
	Denoising too aggressively	Smaller symbols (e.g., articulation markings) missing
Staff processing	Staff not found	Entire staff missing from output
	Staff lines found at incorrect location	Pitch of all notes on single staff incorrect
	Staves not grouped into systems correctly	Some staves associated with wrong instrument, staves in incorrect order
Symbol segmentation	Noise on page identified as symbol	Superfluous accidentals, articulation markings, cluster chords
	Multiple symbols grouped together as one	Symbol omission or misclassification
Symbol classification	Misclassification of accidental	Succeeding pitches incorrect within measure
	Misclassification of clef or key signature	Many consecutive incorrect pitches
	Misclassification of structural element (time signature, barline)	Many subsequent duration errors, depending on how notation is reconstructed
Notation reconstruction	Symbol misassociation	Accidental applied to wrong note in chord
	Symbol not associated with nearby symbol	Ties and slurs that begin but do not end
	Symbol associated with wrong voice, in polyphony	Too much total duration in one voice, not enough in another
Output encoding	Attempt to encode something musically impossible, due to previous errors	Malformed or corrupted output; possible failure to output anything

once, allowing each task to incorporate musical knowledge at earlier points in the recognition process. Jones et al. (2008) described the operation of the commercial OMR software SharpEye as using a "musical-object-candidate-graph" that is slowly populated with primitive shapes as the recognition process proceeds. The graph encodes certain grammatical rules of musical syntax as constraints on its structure, and identified objects are added to the graph as they are identified, not after a full segmentation and identification subtask has been completed. This allows the constraints of the grammar to affect the outcome of previous subtasks as far back as the music symbol isolation step. Chen et al. (2016) used a similar graph-based approach in an OMR system centered around human interaction, ensuring that the symbol identification step is coupled with the notation reconstruction step.

3.1.3 Machine Learning and OMR

A trend in recent research on OMR is the use of machine-learning techniques to perform multiple OMR subtasks simultaneously, reducing the number of discrete tasks in the overall pipeline and therefore the number of possible points of failure.

Binarization of particularly degraded documents is a difficult problem, and the quality of the binarization impacts the quality of the staff-finding and symbol segmentation methods that follow. Calvo-Zaragoza et al. (2017) developed a method to combine these steps into one using convolutional neural networks to separate images of a score into separate layers, each containing only a single type of musical content: staff lines, musical symbols, text, and a layer for the background of the page. This was done by approaching the task as a problem of classifying each pixel on the page, individually, into one of a small number of classes. This work was later extended to use convolutional auto-encoders (Calvo-Zaragoza and Gallego 2019) which can classify large regions of an image at once, substantially speeding up both training and inference. While these methods outperform traditional methods of binarization across a range of manuscript qualities, they require a substantial amount of manually annotated training data and layer separation models trained on one source do not always generalize to others. Since some machine-learning OMR methods do not even require binarized images, other work skips binarization altogether and instead performs layout analysis: locating a bounding box around each individual staff, determining which staves are grouped together into systems, and determining their reading order (Castellanos et al. 2022).

Other work focuses on combining the steps of detecting and categorizing symbols, so that

the step of detecting individual symbols on the page can be informed by what types of musical symbols are expected. Pugin (2006) used hidden Markov models to simultaneously segment and classify the pitch and duration of notes in monophonic historical printed scores. In a similar approach, Baro-Mas (2017) used a bidirectional long short-term memory network (See Section 4.2.5) to process images of monophonic staves, one pixel of columns at a time, into two outputs representing the predicted pitch and predicted duration of the note at each vertical position. Pacha and Calvo-Zaragoza (2018) used techniques from the field of object detection to train a convolutional neural network to simultaneously detect and categorize symbols in monophonic mensural scores, which was successful on a small vocabulary of thirty possible glyphs. Huang et al. (2019) extended this object detection approach to a large corpus of printed scores in polyphonic CWMN. Pacha and Calvo-Zaragoza (2018), Calvo-Zaragoza et al. (2019), and Castellanos et al. (2020) all developed machine-learning architectures that take in unprocessed images of staves of mensural monophonic music and output sequences of identified tokens, skipping many steps of the OMR pipeline at once.

The final two OMR subtasks, reconstruction of musical notation and encoding of music in an output format, can also be replaced by a machine-learning method. Simple monophonic scores can typically be reconstructed into traditional notation without dealing with ambiguities. As the musical surface becomes more involved and contains more semantically important interactions between symbols, rule-based approaches to reconstructing notation are difficult to make robust. Pacha and Calvo-Zaragoza (2019) designed a graph-based format for music notation that encodes notes and events as nodes in a graph and relationships between those notes as different types of edges. They then train a machine-learning model to reconstruct a music notation graph on a set of primitive shapes, and find it to work comparably well to manually tuned rule-based approaches even when the object classification step does not perform perfectly. Further research tackles both notation reconstruction and output encoding in one step, with automatic translation and sequence-to-seqence models (of the kind used to translate between natural languages). These models transform a sequence of primitive shapes directly into a well-formed symbolic music file. This concept was first proposed by Ríos-Vila et al. (2021), who constructed machine-learning architectures based on the Transformer network architecture (Section 4.3.3) for translation of music in agnostic encoding into the Humdrum `**kern` format.

The current state-of-the-art in many areas of OMR is the use of machine learning for end-to-end processing, creating a single model to perform all or nearly all of the OMR subtasks that

are traditionally handled separately. Castellanos et al. (2022) proposed an approach operating on monophonic mensural notation that uses one network to select staff regions on a page and another to convert the images of staff regions directly to semantically encoded musical notation. To extend end-to-end methods past monophony, Alfaro-Contreras et al. (2019) described and evaluated a range of methods for serializing homophonic music into one-dimensional sequences of agnostic tokens, using specialized character codes to signal chords and other simultaneous events. Ríos-Vila et al. (2020) and Alfaro-Contreras et al. (2022) proposed machine-learning methods that transform identified strips of text directly into a semantically encoded output, incorporating the interpretation of agnostic encoding into the network. Some approaches combine the results of multiple OMR systems; Padilla et al. (2014) combined the results of four black-box proprietary OMR applications and was able to produce a result more accurate than any of the systems were individually.

It is difficult to build end-to-end machine-learning architectures that can interpret more structurally complicated musical notation, like pianoform scores (see Figure 3.1d). The degree of interaction between adjacent staves means that some staves cannot be interpreted independently, as is the case with most OMR processes. The task of figuring out the order in which symbols should be read is itself non-trivial; existing approaches that read individual staves from left to right and output sequences of tokens are not suited for this kind of notation. Ríos-Vila et al. (2023) discussed the aspects of pianoform music that make end-to-end OMR difficult and built a machine-learning architecture that reads entire systems left-to-right instead of individual staves.

3.2 Evaluating OMR

Evaluation of OMR, to enable meaningful comparison of different OMR systems, is itself a highly difficult problem. Judging the quality of an OMR system's output requires a method of comparison between two musical scores. The complexity and variety in musical notation, even just within the domain of printed CWMN, makes it difficult to design a single comparison method that is valid for all possible comparisons. Different OMR algorithms are developed specifically for different situations (handling different types of notation, different scan qualities, handwritten vs. printed scores, and so on; see the taxonomy defined in Section 3.1), further complicating the prospect of an apples-to-apples comparison between them.

A systematic review of OMR evaluation methods by Mengarelli et al. (2020) found a wide variety of evaluation methods in use by OMR researchers. Different papers tend to report the results of evaluation on different stages of the OMR process (Section 3.1.2). There is also no standardization in which classification metrics are used, with some using raw accuracy measures, some using more robust measures like the F_1 score, and still others inventing new metrics adapted to their specific task (see Section 2.2.1 for a review of these and other classification metrics). Byrd and Simonsen (2015) noted, however, that OMR is not the only field where evaluation is difficult. Researchers in diverse fields, like speech recognition and text retrieval, have already identified the difficulty of comparing results of different algorithms and addressed it by setting up a standardized, universal format for ground truth, a standardized set of evaluation metrics, and regularly scheduled events where researchers can compare their methods directly. Neither universal standardized formats for ground truth nor standardized evaluation metrics exist for the field of OMR, and it is not practical to include OMR tasks in field-wide competitions like MIREX (Scholz et al. 2016) until such standardized evaluations are agreed upon.

Commercial applications, meanwhile, often self-report evaluation metrics, but without reference to how they were calculated. The marketing material for SmartScore 64 Pro[2] reports "notation, lyrics, and text recognized with up to 99%+ accuracy, depending on print and scan quality," while PhotoScore Ultimate 2020[3] claims to be "over 99.5% accurate on most PDFs and originals." Byrd and Simonsen (2015) surveyed the lack of real evaluation of commercial OMR systems and concluded that it reflects poorly on the state of the field as a whole:

> It is not clear how any of the commercial systems measure accuracy, nor on what music; we have literally never seen an accuracy claim for a commercial system that answered either of these questions. There has been no effective competition between OMR systems on the basis of accuracy, and – as a result – there is far less incentive than there might be for vendors to improve their products, and somewhat less incentive for researchers to come up with new ideas.

The difficulty of producing standard OMR testsets and evaluation metrics has been discussed at length by many other authors as well (Droettboom and Fujinaga 2004; Jones et al. 2008; Byrd and Schindele 2006; Bellini et al. 2007; Byrd and Simonsen 2015; Calvo-Zaragoza et al. 2021). In this section I provide an overview of these issues, with a focus on how previous research has

2. www.musitek.com/smartscore64-pro.html
3. www.neuratron.com/photoscore.htm

defined notions of error between ground-truth correct scores and the results of OMR processes.

3.2.1 Levels of OMR Evaluation

The most acute problem for development of standard OMR evaluation metrics is the hierarchical nature of musical notation. Evaluation metrics for many document analysis tasks, like OCR, can be evaluated by comparing on a per-character basis which characters have been correctly identified using a slate of well-researched performance metrics (Nagy 1995). Musical scores, by contrast, are made up of primitive shapes (see Section 3.1.2) that are grouped together and interpreted into semantically meaningful components in the OMR system's output. An OMR evaluation metric must choose to evaluate on a particular stage of the process: at the *symbol-level* or the *semantic-level*[4].

Droettboom and Fujinaga (2004) noted that semantic-level OMR evaluation is significantly more difficult to define than symbol-level evaluation, as multiple possible semantic encodings of a score can be equally valid, and concluded "that a complete and robust system to evaluate OMR output would be almost as complex and error-prone as an OMR system itself." They proposed comparing the performance of OMR systems by defining ground truth at lower levels, as with their GAMUT symbol-level analysis system. This requires, however, that OMR systems to be compared all use the same vocabulary of primitive shapes to decompose scores. This is far from assured, given the enormous variety of shapes and fonts used in CWMN. Byrd and Schindele (2006) noted that different researchers refer to subtly different stages of the OMR process when discussing symbol-level evaluation; some researchers use "symbol-level" to refer to identified composite (but not semantically interpeted) symbols and others use it to refer to identification of primitive shapes only. The largest OMR-related datasets are appropriate for symbol-level evaluation, including the DeepScores dataset for typeset music (Tuggener et al. 2018), the Music Score Images++ (MUSCIMA++) dataset (Hajič and Pecina 2017) and Handwritten Online Musical Symbols (HOMUS) dataset (Calvo-Zaragoza and Oncina 2014) for handwritten music, and the Universal Music Symbol Collection (Pacha and Eidenberger 2017) for both.

Commercially available OMR systems tend to be "black boxes" that use proprietary OMR processes and only output semantically encoded music files, with little to no information about

4. Most of the cited literature in this section uses the terms "low-level" in place of "symbol-level" and "high-level" in place of "semantic-level". I follow the example of Droettboom and Fujinaga (2004) to use more specific terms, since I have used the terms "low-level" and "high-level" elsewhere in this dissertation.

symbol-level recognition, so any ground truth measure that hopes to evaluate them against other systems must contend at least partially with the difficulty of comparing OMR outputs. One method is to compare semantic-level score files at a symbol-level through manual visual inspection. Bellini et al. (2007) used this methodology in a comparison between three commercial OMR systems, as an attempt at making an evaluation metric that matched expert musician appraisal of OMR output quality. Sapp (2013) provided a detailed measure-by-measure comparison between several popular commercial OMR systems on a single Beethoven piano sonata, focusing on qualitative differences between the outputs and summarizing what kinds of musical phenomena each program struggles at replicating.

Manually inspecting scores output by OMR systems is labor-intensive. To perform automated semantic-level comparison of OMR results with ground truth, there must be some method to align the two to find which parts of the ground truth correspond to which parts of the OMR output. Aligning musical sequences is a difficult task due to the hierarchical nature of semantically encoded music. Byrd and Schindele (2006) noted that even monophonic music is inherently hierarchical and thus nontrivial to align, as accidentals, articulation marks, and grace notes are all usually encoded as "belonging" to some adjacent note or event. MusicDiff (Foscarin et al. 2019) and Mupix (Daigle 2020), discussed more in Section 2.5.2, are both methods of aligning MusicXML files for the purposes of evaluating OMR results. I define a novel way of aligning two polyphonic musical scores as a key step in the error-detection process described in Chapter 5. However, unlike MusicDiff and Mupix, my method first performs a translation from the semantically encoded output of an OMR system to a symbol-level agnostic encoding, in which some details about the semantic structure of the encoded score are discarded, so it is arguably not a true semantic-level comparison method.

There are fewer OMR-related datasets appropriate for semantic-level evaluation than there are for symbol-level evaluation. The largest is the Printed Images of Music Staves (PrIMuS) dataset (Calvo-Zaragoza and Rizo 2018a), which contains only monophonic scores. DeepScores and MUSCIMA++ could potentially be used for semantic-level evaluation for polyphonic music, but they do not provide ground-truth semantic encodings of the musical pieces in the dataset.

3.2.2 Error Weighting and Time-to-Correct

Another difficulty of OMR evaluation, closely related to the symbol- versus semantic-level problem, is that of how to weight the incorrectness of errors relative to one another. It seems sensible

that, for example, omitting an articulation marking is a less destructive error (from the perspective of score readability) than incorrectly predicting the duration of a note, and it would be useful for an OMR evaluation system to reflect that.

A common pattern here, used also in the field of OCR evaluation (Rice et al. 1995), is to attempt to measure the *time-to-correct* or *effort-to-correct* of a given output. Under these evaluations, an output judged to be of poor quality should take longer for a human to correct, or should take more clicks to correct, than one of good quality. The time-to-correct for any particular class of error can be estimated with heuristics (Ng and Jones 2003; Pugin et al. 2007), assessed with human studies by having musical experts judge how much each error impacts the usability of the score (Bellini et al. 2007), or by measuring time-to-correct directly in an interactive correction application (Liu et al. 1999). This usually takes the form of a real-valued weight assigned to each primitive shape or composite symbol; Bellini et al. call this value the *relevance* of each symbol. While it is the most common, time-to-correct is not the only possible weighting scheme for OMR evaluation. Notably, Bugge et al. (2011) proposed a scheme to judge the incorrectness of errors based on how much an OMR error would affect the *sound* of a score if it were performed with that error. This has the effect that, for example, changes in the stem directions or beaming of notes are not counted as errors at all. In its application, however, this error-weighting scheme is implemented similarly to the time-to-correct schemes, differing only in the rationale that decides which types of errors are penalized.

The huge variety in notation styles, notation editors, and the skill level of human correctors complicates the possibility of using time-to-correct or effort-to-correct as a single universal evaluation metric. What is difficult to correct in one editor for one human corrector may be relatively easy in another editor for another corrector, and a catastrophic error in one genre of music may be less important if reproduced in another. Byrd and Simonsen (2015) noted further that the time it takes to correct errors generally depends on the nature and distribution of the errors; for example, if twenty notes in a row have missing staccato markings, in most notation editors it is possible to select that range of twenty notes and apply the markings to all of them. If twenty non-consecutive notes across several different staves are missing staccato markings, the user would have to hunt down each individual one. A time-to-correct evaluation scheme that judges the incorrectness of each error in isolation would assign the same score to these two scenarios.

A related issue concerns the evaluation of errors propagating forward from previous stages of

an OMR process (secondary errors), versus errors made at the stage of the OMR process where evaluation is occurring (primary errors). Because of the highly contextual nature of musical scores, misidentification of a single clef, time signature, or key signature can produce an output where long strings of pitches are wrong or many measures have an incorrect number of beats, even if all the notes in other measures were identified correctly. In extreme cases, when the locations of staves are misidentified or staves are not correctly grouped into systems, the output score can be nearly unrecognizable. If evaluated at a symbol-level, these situations would show the OMR system to be mostly correct; if evaluating on semantic output, they would show the OMR system to have totally failed. Byrd and Schindele (2006) suggest that in OMR evaluation secondary errors should not be counted, which is simple when evaluating systems that allow access to symbol-level information, but more complicated when dealing with the results of commercial black-box applications. This separation in evaluation between primary and secondary errors is not clearly applicable to OMR systems that operate in an end-to-end manner using machine learning, where there may not exist intermediate symbol-level representations at all.

3.2.3 Intrinsic vs. Extrinsic Evaluation

For all information retrieval tasks, a distiction can be made between *extrinsic* and *intrinsic* (or *goal-oriented*) forms of evaluation. Extrinsic evaluation measures how much a retrieval method improves the performance of some larger pipeline, or the performance of a downstream task that takes the retrieval method's output as its input. Intrinsic evaluation, by contrast, attempts to quantify the performance of a retrieval method without reference to a particular use for the results of the retrieval, only comparing the ground truth to the method's results. For example, instead of directly measuring how many errors an OMR system makes (an intrinsic evaluation), one could measure how much the OMR system can shorten the time it takes a human to digitize and transcribe a printed score (an extrinsic evaluation). Extrinsic evaluation does not have to involve human interaction; one could evaluate the use of an OMR method as a part of a larger music information retrieval pipeline that itself is totally automatic.

Trier and Jain (1995) discussed the benefits of extrinsic evaluation in the context of document analysis tasks. The main benefit to extrinsic evaluation is the relative ease of interpretation of the results; if a downstream task has a well-understood and standardized method for measuring its performance, then that well-definedness can extend to the extrinsic evaluation of the whole system. Some works have explicitly employed extrinsic evaluation of OMR, such as Balke et

al. (2015), who tested the uncorrected results of an OMR system on melodic matching tasks. deGroot-Maggetti et al. (2020) evaluated the performance of several retrieval algorithms that operate on symbolic music, like key-finding and harmonic analysis, when given uncorrected or partially corrected scores from an OMR process as input. This extrinsic evaluation assumed that an OMR process could be judged on the basis of its usefulness for large-scale computational analysis of musical scores. The "World Cup of OMR" competition performed by Daigle (2020) was a study that compared how quickly expert transcribers could transcribe large polyphonic scores from scratch against how quickly they could correct the same scores output from commercial OMR systems. Effort-to-correct metrics (Section 3.2.2) may also be considered as forms of extrinsic evaluation. Extrinsic evaluation has seen more widespread use in general document analysis research than in OMR. As part of an OCR workflow, Obafemi-Ajayi and Agam (2013) developed a method for evaluating binarization algorithms based on how they affected performance of page segmentation tasks that are run directly after the binarization step. Similarly, Stamatopoulos et al. (2015) evaluated page segmentation tasks based on how they affected the performance of subsequent text-line segmentation tasks.

Hajič (2018) argued for the development of intrinsic evaluation metrics for OMR tasks, and cites two major reasons for its necessity. First, that extrinsic evaluation is typically expensive. Many forms of extrinsic evaluation, such as Daigle's World Cup of OMR, are user studies that require paying expert musicians to perform long, laborious tasks. The research field of OMR is too small to practically support enough user studies to compare all the systems being developed. Second is that extrinsic evaluation of OMR results are generally more difficult to use when comparing the performance of different OMR evaluation studies. Metrics such as effort-to-correct are only comparable across trials that involve people using a particular notation software, with commensurate levels of experience in that software, all correcting OMR output on similar scores. If an intrinsic evaluation method were developed, then new OMR systems could be cheaply and easily compared to one another without repeating expensive user studies.

Another problem with extrinsic evaluations of OMR is that so far in the field's history, evaluation methods closer to the symbol level have received more attention and standardization than those closer to the semantic level. There exist some individual stages of the OMR process that have established datasets and reasonably standardized recognition metrics, such as classification of isolated musical symbols (Pacha and Eidenberger 2017). Semantic-level evaluation, by contrast, has recieved less attention. This is the opposite of an ideal situation for the use of ex-

trinsic evaluation, as the downstream task (assembling a semantically encoded score) is harder to meaningfully evaluate than the tasks required before it (symbol recognition, staff-finding, etc.). To address these problems and promote the use of intrinsic OMR evaluation, Hajič proposed the development of a notion of musical edit distance (see Section 2.5.2) definable on all musical scores, and not tied to a single music notation format like MusicXML whose standards tend to change over time.

3.3 Photoscore

The aim of the error detection method I describe in Chapter 1 is to be capable of detecting errors in the output of a particular OMR system. I will evaluate this methodology using PhotoScore version 8.0.4[5], released in 2015.

Multiple factors motivated my choice of this particular OMR software. I considered using an open-source OMR system like Audiveris[6], which would have allowed access to more information about the OMR process that could be incorporated in the error detecting step; for example, confidence measures on the classification of each symbol. However, I decided that I wanted the method to be capable of operating on proprietary black-box OMR methods, which are far more widely used by musicians than open-source solutions. There exist multiple other modern proprietary OMR systems that have MusicXML export functionality and so would have been appropriate for this research, including SmartScore[7], ScanScore[8], PDFtoMusic Pro[9], capella-scan[10], and PlayScore 2[11]. As discussed in Section 3.2, there has been little research directly comparing the capabilities and accuracies of these competing commercial systems that I could use to justify a choice of one application over another.

In the absence of concrete data regarding the performance metrics of various OMR applications, my decision to utilize PhotoScore is founded on two primary considerations. Firstly, PhotoScore's relative popularity influenced my choice. Although there are no specific data detailing the market share of each OMR system, the term "PhotoScore" registers higher search query volumes on Google compared to other OMR systems[12]. Furthermore, PhotoScore is pack-

5. https://www.neuratron.com/photoscore.htm
6. audiveris.github.io/audiveris/
7. www.musitek.com
8. scan-score.com
9. www.myriad-online.com/en/products/pdftomusicpro.htm
10. www.capella-software.com/us/index.cfm/products/capella-scan/info-capella-scan/
11. www.playscore.co
12. This judgment was made using the Google Trends interface to compare the relative popularity of each

aged with and integrated into Sibelius, an industry-leading music notation application[13]. An error detector designed to work on PhotoScore's OMR outputs is likely to have the broadest impact and utility for musicians and music researchers, given its extensive user base.

My personal familiarity with PhotoScore also plays a significant role in this decision. Having an understanding of the types of errors PhotoScore typically produces allows me to more effectively evaluate the error detector's performance. The Mendelssohn String Quartet Dataset (MSQD), which I participated in creating, began as a project in 2019 using Photoscore 8.0.4 (deGroot-Maggetti et al. 2020); even before then, I collaborated with others who were using this particular version of Photoscore for their OMR-related projects, so I am most familiar with this particular version. While the latest version available as of writing this dissertation is PhotoScore Ultimate 2020, this research began prior to its release, so I continue using the version I initially started with.

As proprietary software, PhotoScore does not make public how it approaches OMR; their marketing material frequently mentions a "OmniScore Dual-Engine Recognition System," as in the documentation describing its integration into the Sibelius notation application, but there is no further mention of this system other than that it is the component of Photoscore that reads music. Still, it is possible to get a sense of how it performs each of the traditional OMR subtasks through use of the program and the user guide for the program that is distributed with the software.

3.3.1 PhotoScore's OMR Process

In this Section I describe the OMR process of PhotoScore as precisely as I can with the limited information available. The objective of this analysis is to catalog the potential failure cases of PhotoScore's OMR process, in order to inform and justify the design of an error detector that will be built to operate on its outputs. It is important to emphasize, however, that the overarching goal remains to develop an error detector capable of being trained on the outputs of *any* proprietary, black-box OMR system. Consequently, the design choices made will not rely on the assumption that the output is exclusively from PhotoScore. In analyzing PhotoScore's errors, I treat them as a representative subset of all potential OMR errors. This implies that the detector should be capable of identifying *at least* the errors that appear to PhotoScore's outputs.

term over time. Visit the URL `trends.google.com/trends/explore?date=all&q=PhotoScore,SmartScore` for an example of such a comparison.

13. `www.avid.com/products/photoscore-and-notateme-ultimate`

Considerations regarding the broader applicability of the error detector to other OMR systems, including those with different error profiles, will be explored in Chapter 7. This approach ensures that while the immediate focus is on PhotoScore, the methodology and insights gained can provide a foundation for extending the detector's capabilities to other OMR systems.

PhotoScore offers a custom interface for connecting to scanning hardware and taking scans of documents, including options for cropping pages, choosing resolutions, rotating individual pages to line staves up with the bounds of the file, and choosing a scanning resolution. I focus on the analysis of documents that have already been previously scanned, so I will omit a detailed description of this segment of the program and instead describe the workflow resulting from importing scanned Portable Document Format (PDF) files. Upon importing PDF files to PhotoScore and selecting which pages of the input PDF to process, the program quickly reveals two dialog boxes in sequence: one with the message "Converting to black and white," and one with the message "Cleaning up." It is likely that these correspond to some image binarization step and some despeckling or noise removal on the binarized image. After all images have been binarized and cleaned up, they are added to a queue of scanned images, and the OMR process is performed on each one of them, in the background, without user input. As the OMR process completes, each page is individually removed from the queue and placed in a list labeled "Formatted Scores." The user can then review the results of the OMR and compare it with the original. When the user hovers the cursor over a point on the view of the OMR'ed score, the upper portion of the window shows the corresponding point on the same staff in the input image.

The program does expose one important aspect of its OMR process to the user: detection, classification, and grouping of staves on the page. After the OMR process has already been run on a page, the user can inspect where PhotoScore found staves on each page and how it grouped them together into systems. The user can then correct the results by scaling and rotating the staves and manually selecting how they are to be grouped into systems, after which the page is sent back to PhotoScore for the symbols on each staff to be processed again using this new information. Figure 3.6 shows how the PhotoScore staff-finding interface looks during a failure case of the staff-finding algorithm. Here, the uppermost staff corresponding to the first violin part has not been recognized at all. Note how on the second system of the page, there is no red line linking the left edges of each of the four parts together, as there is in the first system; this signals that PhotoScore has not identified these staves as belonging to the same system, due to

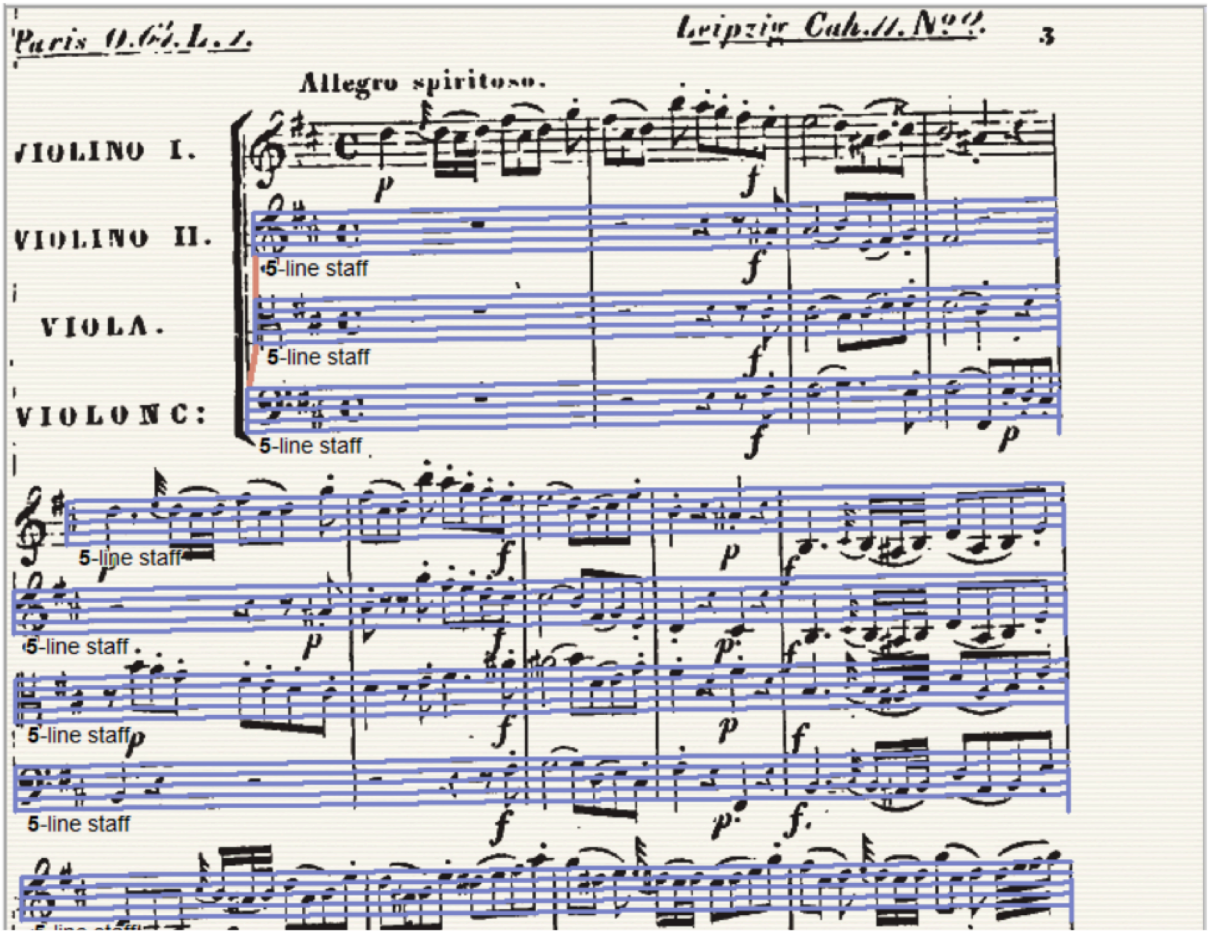
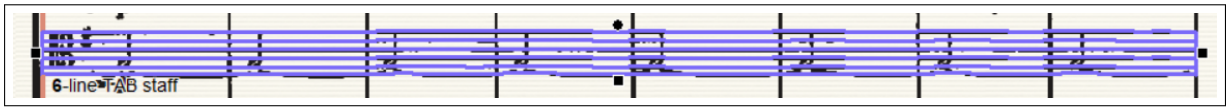


Figure 3.6: An image of PhotoScore's staff-finding interface, failing on a scan of the first page of Joseph Haydn's String Quartet in B Minor, Hob. 68, Op. 64 No. 2.

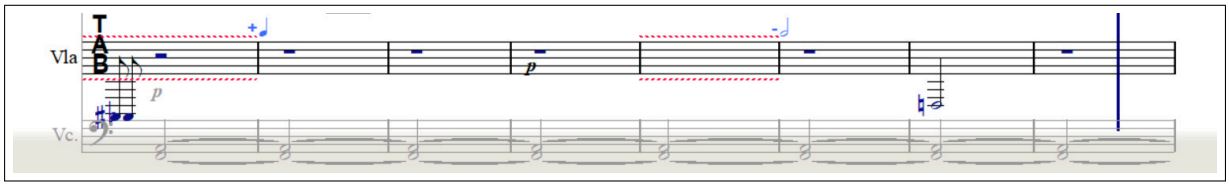
the left border of the system being cut off the side of the scan. As a result, PhotoScore assumes that each of these individual lines are to be played by the same instrument, in order from top to bottom.

Figure 3.7 shows another possible failure case: here, a long series of tied half notes are engraved in a style that makes the ties appear nearly parallel to the staff lines in parts, and so PhotoScore has interpreted them as another staff line. Since it has found six staff lines parallel to one another it assumes the staff must be a six-line guitar tablature staff. This leads to the staff being output as a five-line staff with a tablature "clef" containing ordinary notes that stretch down to the staff below (Figure 3.7c). Exporting this result to a MusicXML file and opening it in most music notation editors will result in a notification that the file is corrupted, and either fail to open it or perform some heuristic procedure to try to fix the file. This particular kind of error, where a staff is mis-classified as a staff with a different number of lines, happens relatively often on degraded historical prints when the staff lines are faint or broken.

(a) The original scan of the viola part snippet.



(b) The results of PhotoScore's automatic staff detection on the snippet above.



(c) How the snippet appears in PhotoScore's correction interface after it attempts to find musical symbols on it.

Figure 3.7: String Quartet No. 1 by Leoš Janáček, "Kreutzer Sonata", mm. 14–21, Viola part, at various points through PhotoScore's OMR process.

3.3.2 Correction Interface

After staff lines have been corrected to the user's satisfaction, PhotoScore provides a Graphical User Interface (GUI) for the user to review and correct its output. This is the particular step of the OMR process that my error detector seeks to speed up, so I will describe it in some detail.

PhotoScore provides functionality standard to most music notation editors. Users can make corrections by selecting, deleting, or moving symbols on the page, or by adding new ones from a keypad of common musical symbols. Larger structural elements like clefs and time signatures, and spanning elements such as slurs, ties, and hairpins, are modified via a right-click context menu. At this stage, altering the shape and layout of staves or reorganizing staves into different systems is not possible. To make such changes, users must revert to the staff adjustment interface, modify as needed, and then re-run OMR for the entire page. This action resets any prior corrections made.

A key feature of PhotoScore's correction GUI is its synchronized view between the original document and the OMR output (illustrated in Figure 3.8, labels (c) and (d)). Hovering the cursor over any element in the OMR output displays the corresponding area in the input image, facilitating a clear comparison. This feature significantly aids users in verifying the OMR results against the original scan. The main way that users can correct the score is by selecting and deleting or moving symbols on the page, or by adding new ones by selecting them from a keypad of common symbols. Changing larger structural elements like clefs and time signatures, as well

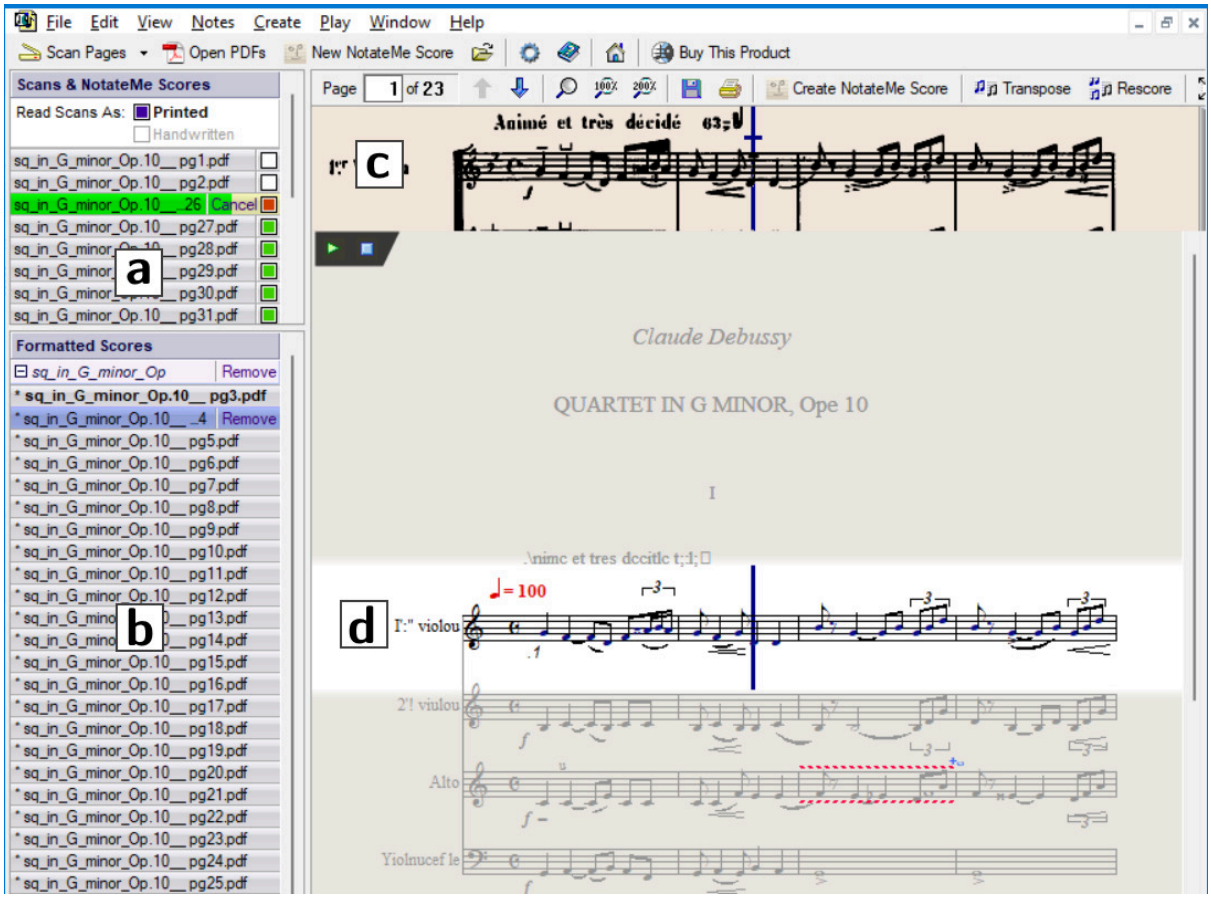


Figure 3.8: An image of the GUI of PhotoScore 8, running on Windows 10. Four main components of its user interface are labeled: (a) the list of pages yet to be run thorugh PhotoScore's OMR engine, (b) the list of pages already recognized, (c) a view on the binarized version of the currently selected recognized page and (d) the results of the OMR process on the currently selected page.

as managing spanning elements like slurs, ties, and hairpins, is done using a right-click context menu.

The other major tool that PhotoScore supplies is a utility for identifying mismatches between durations and time signatures, referred to by the program as the "bad timing navigator." This function evaluates each measure to ensure the total durations of notes match the expected duration defined by the prevailing time signature. In polyphonic music, this check is conducted separately for each voice, with discrepancies tracked individually for each. An illustration of the bad timing navigator's functionality after OMR on a manuscript is provided in Figure 3.9. Within the OMR correction interface, any measure identified with incorrect timing displays an annotation at the top-right corner indicating the total duration value of the discrepancy. For instance, if a measure in a $\frac{4}{4}$ time signature contains only three beats' worth of notes, PhotoScore would display the symbol -♩ to denote a shortfall of a quarter note in that measure. This approach

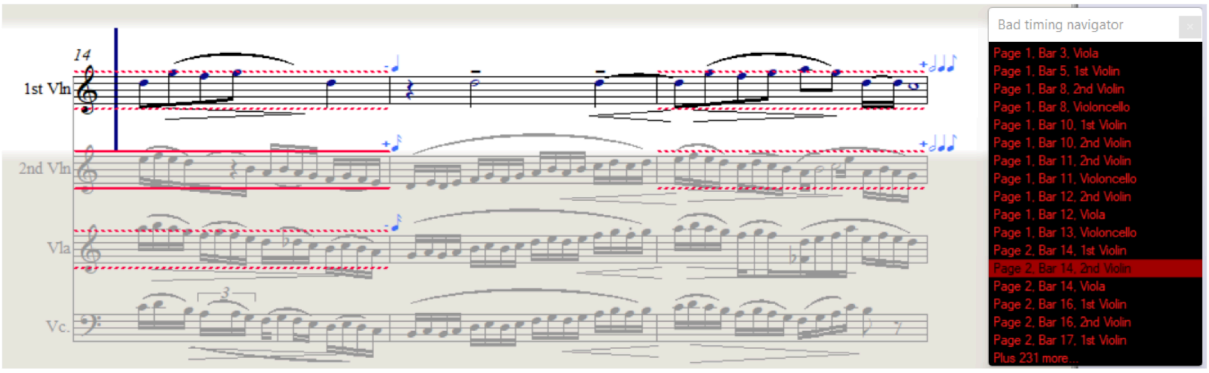


Figure 3.9: PhotoScore's bad timing navigator tool on Claude Debussy's String Quartet no. 1, Op 10, mm 14-16. Note that the time signature of this snippet of music is $\frac{4}{4}$.

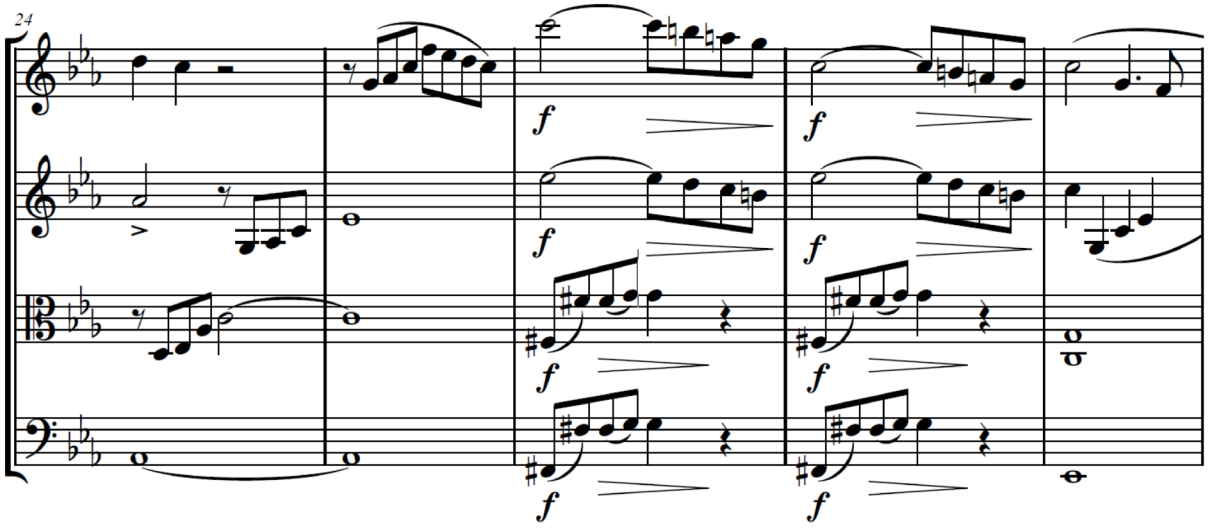
to detecting metrical errors is a common feature in OMR systems. A more comprehensive discussion on similar methodologies employed by other OMR systems, particularly as a post-processing action, is explored in Section 2.5.1.

3.3.3 Example PhotoScore Outputs

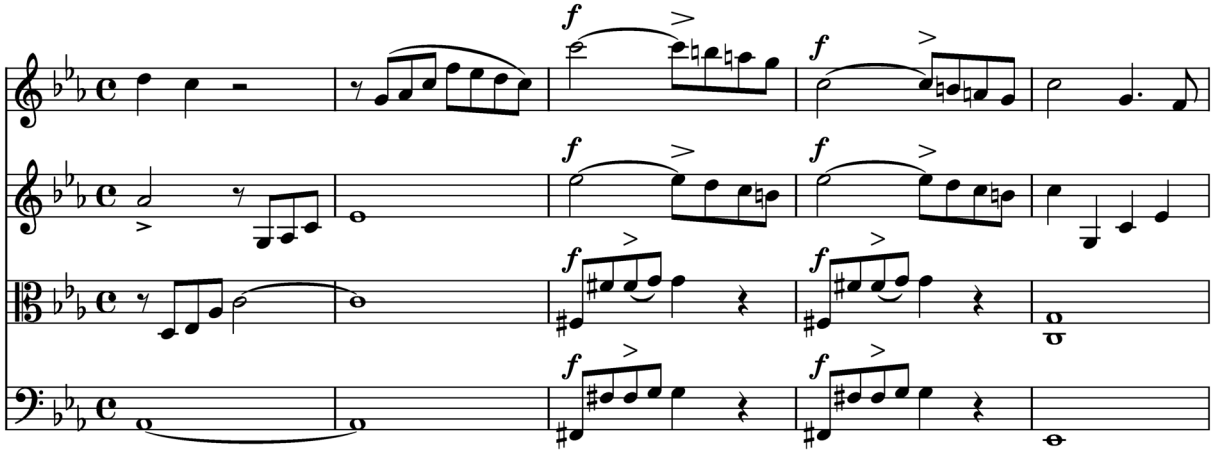
This section showcases examples of musical score images alongside their unedited outputs from PhotoScore 8.0.4's OMR system. These examples demonstrate the types of musical phenomena that my error detection method must address and how the system performs on scans of varying quality. This is not an exhaustive evaluation of PhotoScore's OMR capabilities, as the program is designed for some level of user interaction, which I have bypassed in these examples. For clarity and readability, the outputs are presented using MuseScore 4.0.2, rather than as they appear PhotoScore's interface. This is because PhotoScore's typesetting and engraving capabilities are limited, making it difficult to format a score to be readable in the context of this document. All scores used are public domain scans sourced from the International Music Score Library Project (IMSLP) in PDF format.

Figure 3.10 depicts a five-measure excerpt from Fanny Hensel's (1805-1847) String Quartet in E-flat Major, composed in 1834 and digitally engraved by members of the OpenScore String Quartet project (Gotham et al. 2023). In such digitally rendered score images, PhotoScore commits few errors. The primary inaccuracies pertain to markings spanning multiple notes. For instance, slurs and ties often begin and end at incorrect locations, or are entirely omitted (as seen in the viola and cello parts, measures 3-4). Additionally, across all voices the original score's hairpins are incorrectly converted into accent markings on individual notes.

Figure 3.11 presents a five-measure segment from a densely notated section in Robert Schu-



(a) A scanned image of the Hensel string quartet excerpt.



(b) The results of PhotoScore's OMR on the string quartet excerpt.

Figure 3.10: Fanny Hensel's String Quartet in E-flat Major, Mvt. 1, mm. 24–29.

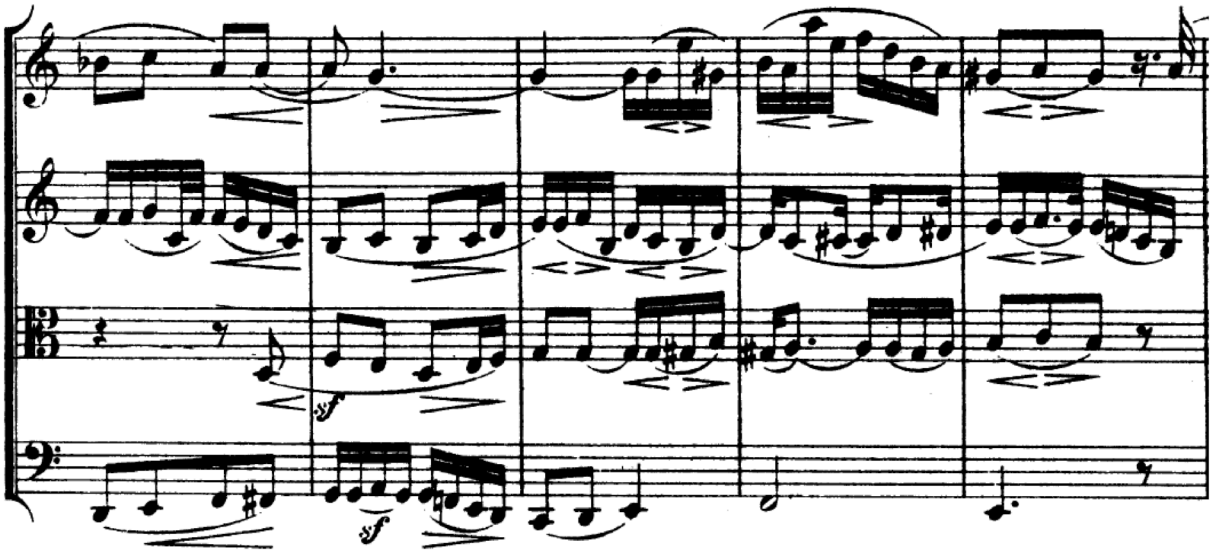
mann's (1810-1856) String Quartet Op. 41, No. 1, composed in 1842. This image is a high-quality scan of the Breitkopf & Härtel edition dating from 1881, available on IMSLP[14]. Even in dense sections, PhotoScore generally accurately locates all note heads. However, issues arise in the form of incorrect barring (seen in the 2nd violin, measure 4), omitted accidentals (1st violin, measure 3 and cello, measure 2), and entirely missing notes (1st violin, measure 5). Such errors are more prevalent in areas where musical symbols are densely clustered, particularly when these symbols are in close proximity or overlapping. This tendency highlights a common challenge in OMR processes when dealing with complex musical textures and closely spaced notational elements.

Figure 3.12 shows a five-measure excerpt of Felix Mendellsohn's (1809-1847) String Quartet in A-flat Major, Op.44, No. iii, composed in 1838. This excerpt originates from a lower-resolution scan of the Breitkopf & Härtel publicatioon dating from 1875, available on IMSLP[15]. This example is particularly illustrative of PhotoScore's performance in less-than-ideal conditions. Noticeable are the broken staff lines, the very faint stems of notes, and the blurred beams in runs of sixteenth notes. While many note heads are correctly identified by PhotoScore, a significant number are entirely missing. When PhotoScore cannot decide on a valid interpretation of the symbols it detects, it usually opts to fill the relevant section of the staff with rests. Occasionally, as in the first measure of the 2nd violin part, it erroneously deduces the necessity for two polyphonic voices on a single staff, resulting in a string of rests filling the gaps in the less populated voice. This scan represents the lower threshold of quality that PhotoScore can process while still generating an output resembling the original score. Scans with more smudging or distortion typically lead to failures in staff identification, staff system grouping, or many measures being indiscriminately filled with rests.

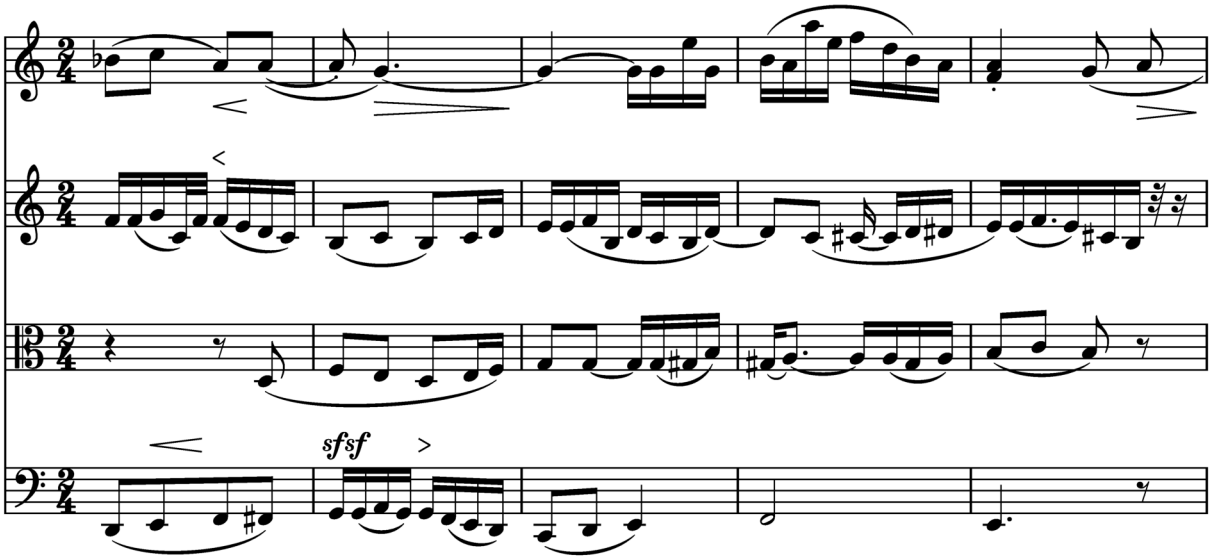
3.4 Modeling User Interaction in the OMR Correction Process

In this section, I assess the potential time savings offered by the error detector in the process of correcting OMR output. To accomplish this, it is necessary to focus on the aspect of the process that the detector influences: identifying errors. The detector is designed to streamline the error-finding step in the correction process but does not actually speed up the correction of errors that the user has already identified. Therefore, evaluating the efficacy of the detector

14. `imslp.org/wiki/String_Quartet_No.1%2C_Op.41_No.1_(Schumann%2C_Robert)`
15. `imslp.org/wiki/String_Quartet_No.5%2C_Op.44_No.3_(Mendelssohn%2C_Felix)`

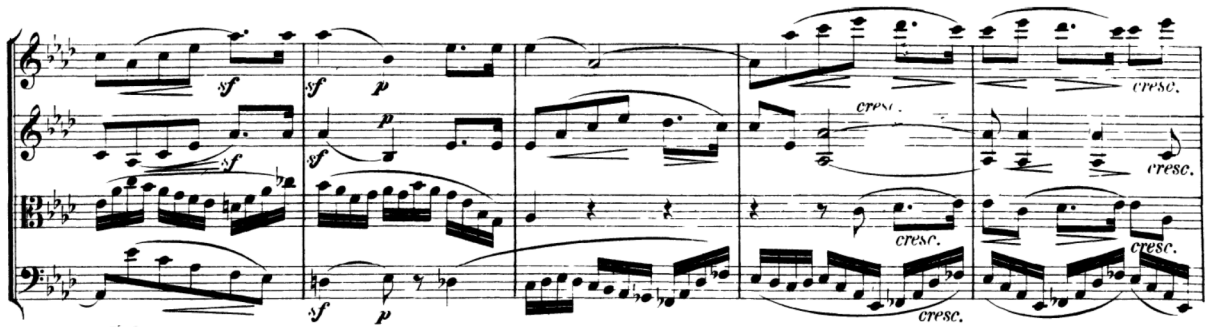


(a) A scanned image of the Schumann string quartet excerpt.

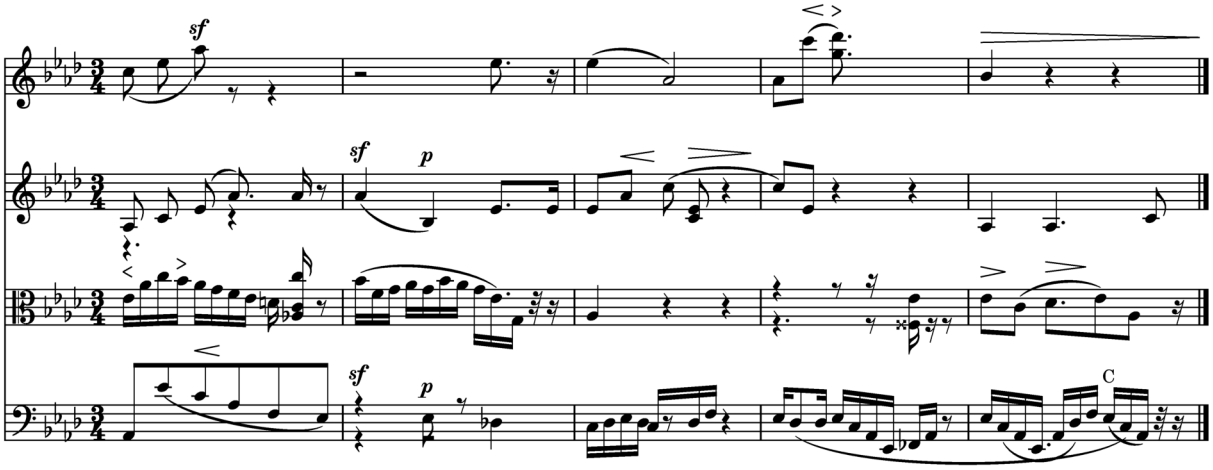


(b) The results of PhotoScore's OMR on the string quartet excerpt.

Figure 3.11: Robert Schumann's String Quartet Op.41 No.1, Mvt. 1, mm. 8–14.



(a) A scanned image of the Mendelssohn string quartet excerpt.



(b) The results of PhotoScore's OMR on the string quartet excerpt.

Figure 3.12: Mendelssohn's String Quartet in A-flat Major, Op.44-iii, Mvt. 4, mm. 101–106.

involves analyzing the proportion of time users spend comparing the original scan with the OMR output to identify errors.

Conducting a comprehensive user study would be the most direct way to measure how much time could be saved by the error detector. Such a study would involve having participants correct OMR outputs with and without the aid of the error detector and measuring the time taken in both scenarios. However, such studies are expensive, requiring the expertise, time, and effort of professional music engravers. Moreover, integrating the error detector into modern notation applications in a user-friendly manner, akin to PhotoScore's correction interface (illustrated in Figure 3.8), would be necessary. For these reasons, I consider a user study to be beyond the scope of this dissertation; for a discussion on potential future user trials, refer to Section 7.3.6.

In lieu of a user study, I propose an alternative method to estimate time savings from using the error detector. This method involves calculating the proportion of the correction process time dedicated to finding errors in the OMR output. By understanding this value, it is possible to estimate how much time could potentially be saved if the error detector reduces the time spent in this error-finding phase.

3.4.1 The Keystroke-Level Model of User Interaction

I propose to use the Keystroke-Level Model (KLM) of user interaction to estimate the potential time savings offered by the error detector during OMR correction processes. The KLM conceptualizes user interactions with a GUI as sequences of distinct actions, categorized into a limited set of operations. This model, initially proposed by Card et al. (1980), is application-agnostic and deconstructs human-computer interactions into sequences of basic operations. A common version of the KLM includes the following operations and their corresponding abbreviations:

- **K**: Button presses or keystrokes.
- **P**: Moving the cursor to a target on a display with a mouse.
- **A**: Referencing another display for instructions on what task to perform next.
- **S**: Visually searching the display for a particular target.[16]
- **H**: Homing the hands on the keyboard (returning to a neutral position).

16. The **S** and **A** operations were not originally present in the original formulation of KLM, instead grouped together with the **M** operation. They were introduced as separate operations by Roberts and Moran (1983) and are useful in this context.

- **D**: Drawing straight lines with the cursor (e.g., when selecting text).
- **M**: Mentally preparing or planning to execute a particular action.
- **R**: Waiting for the system's response.

Each operation within the KLM is associated with formulas for heuristic time estimates, derived from extensive user testing. These heuristics provide an idea of the duration each operation should take, factoring in variables such as the user's proficiency with the specific software. By breaking down a task into a sequence of these operations, one can estimate the time required for a user to complete the task within a particular GUI. The KLM has been refined and expanded by various researchers (Al-Megren et al. 2018), adapting it for different types of devices (like tablets or smartphones) and software applications (such as text editors, graphic design tools, and web browsers). A notable adaptation by Nowakowski and Hadjakos (2023) tailored the KLM for music notation editors, building upon earlier work by Roberts and Moran (1983) that focused on user interaction in text editors. KLM is frequently employed as a predictive tool in GUI design, providing a way to estimate user task completion times before the actual implementation of the interface. When applied correctly, KLM can predict task completion times with a high degree of accuracy, typically within 10–30% of the actual measured times (Sauro 2009).

To analyze OMR correction within the KLM framework, I base the task on a scenario where the OMR output and the original scan are viewed side-by-side on a single display. This setup leads to a proposed sequence of operations that categorize each step of the OMR correction process. Research suggests that the average perceptual span for reading music in CWMN is between two and four beats per glance (Madell and Hébert 2008). Based on this, it is assumed that the corrector works through the score one measure at a time.

The sequence of operations for OMR correction is as follows:

1. **A** (Gather task information): Observe a measure in the original scan.
2. **S** (Search): Examine the corresponding measure in the OMR output to identify visual differences. If no differences are found, move to the next measure and return to Step 1.
3. **M** (Mental): If discrepancies are identified, analyze the semantic differences: what specific musical elements require correction?

4. **M** (Mental): Determine the most efficient correction method within the constraints of the notation editor. For instance, assess whether the issue can be resolved with a few keystrokes or if it would be quicker to delete and rewrite the entire measure.

5. Several **K** (Keystroke) and **P** (Pointing) operations: Perform the necessary mouse movements and keystrokes to rectify the error. After completing the correction, proceed to the next measure and return to Step 1.

Since an error detector only provides annotations and extra information on top of the OMR output, it can only reduce the amount of time spent in the first two steps in this process that involve searching for tasks to complete. I refer to these first two steps, where the user searches for discrepancies between the OMR output and the original scan, as the *task-finding* period, to roughly match terminology used in KLM research. Crucially, when a user finds no differences between a measure in the scan and the OMR output, task-finding constitutes the entirety of the task time for that measure, as the corrector then immediately moves on to the next measure. As a result, the proportion of time devoted to task-finding increases as the number of errors present in the score decreases.

For an error detector to significantly enhance the efficiency of the OMR correction process, three conditions must be met. First, the error detector must effectively reduce the time spent in locating errors that require correction. Second, the task-finding phase should represent a substantial portion of the total time spent on correction; that is, correcting an error must not take much longer than identifying it. Third, the OMR output should not contain a high proportion of errors. In other words, if the majority of measures contain errors, the corrector will still need to scrutinize nearly every measure, eliminating the time-saving advantage of the error detector.

To quantify the potential time savings offered by an error detector in the OMR correction process, I propose a formula that captures the key variables involved.

- t_{compare}: The average time necessary to visually compare a measure[17] from the original scan with the corresponding measure in the OMR output. This corresponds to **A** and **S** operations in the KLM framework.

- t_{correct}: The average time necessary to perform the operations necessary to correct a mea-

17. The term "measure" is used here for illustration, but the formula would be equally applicable to any definable unit of music notation, such as half a measure, two measures, or a specific number of symbols.

sure once the presence of an error has been identified. This corresponds to **M**, **K**, and **P** operations in the KLM framework.

- t_{proof}: The average time necessary to proof a measure that has errors, defined such that $t_{\text{proof}} = t_{\text{compare}} + t_{\text{correct}}$.
- N: The total number of measures in a score.
- T: The total time necessary to correct a piece of OMR output.
- c: The proportion of the measures in a score that contain errors and require correction, $0 \leq c \leq 1$.

Then, the total time necessary to correct a score of OMR output is

$$\begin{aligned} T &= N((c-1)t_{\text{compare}} + ct_{\text{proof}}) \\ &= N((c-1)t_{\text{compare}} + c(t_{\text{compare}} + t_{\text{correct}})). \end{aligned} \tag{3.1}$$

Next, define p as the proportion of the total proofing time spent on comparing measures:

$$\begin{aligned} p &= \frac{t_{\text{compare}}}{t_{\text{proof}}} \\ &= \frac{t_{\text{compare}}}{t_{\text{compare}} + t_{\text{correct}}}. \end{aligned} \tag{3.2}$$

Equation 3.1 can be rewritten using p:

$$T = Nt_{\text{proof}}(p(1-c) + c). \tag{3.3}$$

Now, consider an error detector capable of accurately identifying a certain proportion of the score as erroneous, with a near-perfect recall rate. Let e represent this proportion, with $0 \leq e \leq 1$. This means the detector reliably indicates that no errors are present outside the marked areas. With the use of such an error detector, the human corrector would not need to examine the unmarked portions of the score. The revised time necessary to correct the score T_{detector}, factoring in the assistance of the error detector, becomes:

$$T_{\text{detector}} = Nt_{\text{proof}}(ep(1-c) + c). \tag{3.4}$$

Dividing Equation 3.4 by Equation 3.3 gives an expression for $\frac{T_{\text{detector}}}{T}$. This statistic describes how long it takes to correct a score using the error detector as a proportion of the time it would

take to correct the score without the error detector.

$$\frac{T_{\text{detector}}}{T} = \frac{ep(1-c)+c}{p(1-c)+c}. \tag{3.5}$$

Finally, I define d, which defines the potential time savings of the error detector as a proportion of the total time without the detector, in terms of e, p, and c:

$$\begin{aligned} d &= 1 - \frac{T_{\text{detector}}}{T} \\ &= 1 - \frac{ep(1-c)+c}{p(1-c)+c}. \end{aligned} \tag{3.6}$$

Equations 3.5 and 3.6 are only valid when $e \geq c$, as if the error detector finds fewer errors than exist in the score, then its recall cannot be near-perfect, which was assumed. To use this formula to derive estimates for how much time the error detector could save, it is necessary to come up with an estimate for p suitable for use in a KLM of music notation editor interaction.

3.4.2 Lower and Upper Bounds on Time Spent Searching for OMR Errors

In the context of OMR correction, determining the duration that users spend locating errors is a difficult task. To my knowledge, there is no specific research reporting the time taken by human transcribers to compare musical scores against ground truth. Studies on sheet music reading primarily focus on music performance, which significantly differs from reading music in an analytical context, making it challenging to draw relevant conclusions for OMR correction (Madell and Hébert 2008). While error detectors for natural language are extensively studied, their evaluation typically centers around retrieval metrics like precision and recall, rather than how much time they can save (see Section 2.2 for an overview).

However, insights can be gleaned from related research in text editing. Roberts and Moran (1983) recorded detailed keystroke-level metrics for various text editing tasks, providing a framework that can be adapted to estimate the time spent in task-finding during OMR correction. In this study, participants executed "core tasks" like insertion, deletion, replacement, or movement of elements (characters, words, sentences, paragraphs, etc.) in a text editor. These tasks were broken down into KLM subtasks: referencing a document for instructions, locating the task area in the editor, mentally preparing for the task, and executing the task through mouse movements and keystrokes. The initial two steps align closely with the task-finding phase in OMR correction. The study found that referencing the instruction document took a median of 2.0 seconds,

while visually locating the task area in the editor took 2.6 seconds, totaling 4.6 seconds. Given that the median total task time was 19.2 seconds, approximately 24% of the user's time was devoted to searching for information across two displays. Therefore, in this context, the value of p (the proportion of time spent in task-finding) was 0.24. The task-finding setup in this study, requiring the user to cross-reference information from two documents, is comparable to the task-finding process of OMR correction, so these insights can be generalized to the current context.

Estimating the value of p for music notation editing involves understanding the average duration required for a user to correct an error once it is identified. A relevant study by Nowakowski and Hadjakos (2023) used KLM to estimate the time necessary to complete various core tasks in music notation editors. These tasks, reflecting the methodology of Roberts and Moran, included operations such as adding, deleting, replacing, and moving different musical objects (notes, rests, articulation marks, chords, measures, parts, or systems). Unlike the earlier study, Nowakowski and Hadjakos did not model a task-finding period. Their findings suggest that core tasks in music notation editors are completed more quickly on average than those in text editors, with median interaction times of 4.9 seconds using a mouse and 3.7 seconds using a keyboard. This difference is likely because Nowakowski and Hadjakos modeled a skilled transcriber using software they were familiar with, while Roberts and Moran used subjects with a range of experience levels in text editing, on text editors that they had not used before.

The final estimation necessary is to conjecture how many core tasks, as defined by Nowakowski and Hadjakos, are typically needed to correct a measure in an OMR-processed score. This figure varies considerably, as some OMR errors might be fixed with a single keystroke (e.g., adding a note or deleting an accidental), while others may necessitate more complex edits, such as repositioning a group of notes or retranscribing a measure entirely. Determining the exact number of core tasks required to correct an OMR output would demand a method for converting differences between scores into an optimal list of core tasks tailored to a specific notation editor and input method. Currently, there is no research providing such a detailed analysis. For the purpose of this discussion, I propose both conservative and optimistic estimates for the average number of core tasks needed to correct a measure in music notation editing. These estimates are informed by the observed speeds of expert music transcribers using keyboard input.

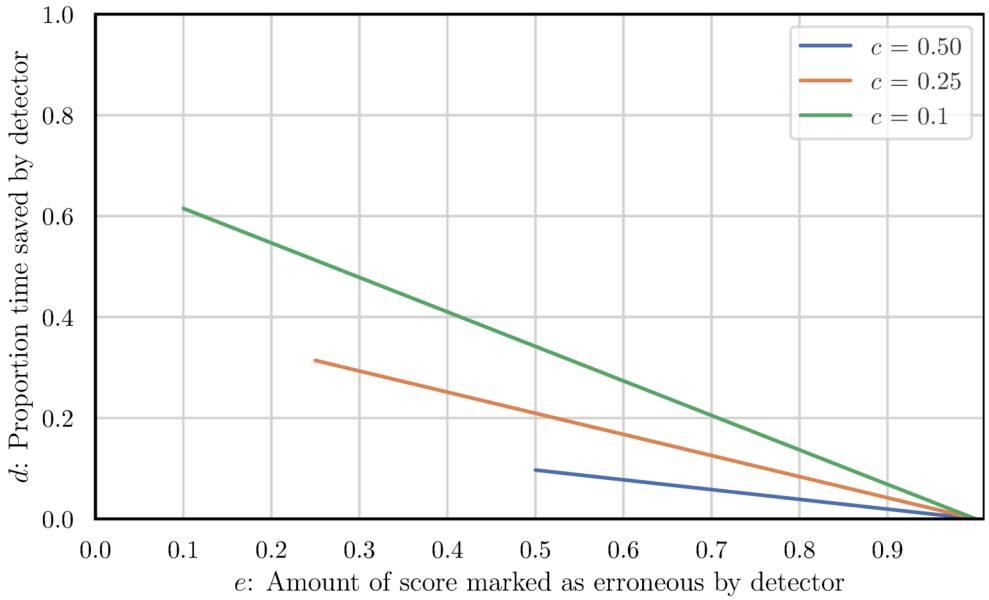
For the conservative estimate, it is assumed that correcting a measure of OMR output will not exceed five core tasks for a skilled user. Given that proficient users can enter pitches using

keyboard shortcuts almost as quickly as typists input text (Daigle 2020), most measures can be retranscribed from scratch within this limit. The time required for five core tasks using keyboard shortcuts is calculated as $3.7 \times 5 = 18.5$ seconds. To err on the side of caution, this figure is rounded up to 19.2 seconds, aligning with the time estimated by Roberts and Moran (1983). This approach equates the time it takes expert transcribers to correct a measure with the time unskilled typists need to complete a core task in an unfamiliar text editor. Hence, the lower bound for p is set at 0.24, the same proportion that was derived from Nowakowski and Hadjakos's study. For the optimistic estimate, the fastest median core task time is taken into account, which is 3.2 seconds, achieved using the editor MuseScore 4 and keyboard input. Assuming an ideal case where every error in a piece of OMR output can be corrected with just one core task, and assuming that task-finding takes 4.6 seconds as previously estimated, the total task time becomes $4.6 + 3.2 = 7.8$ seconds. In this scenario, task-finding constitutes approximately 59% of the total task time. Thus, 0.59 is considered the upper bound for p.

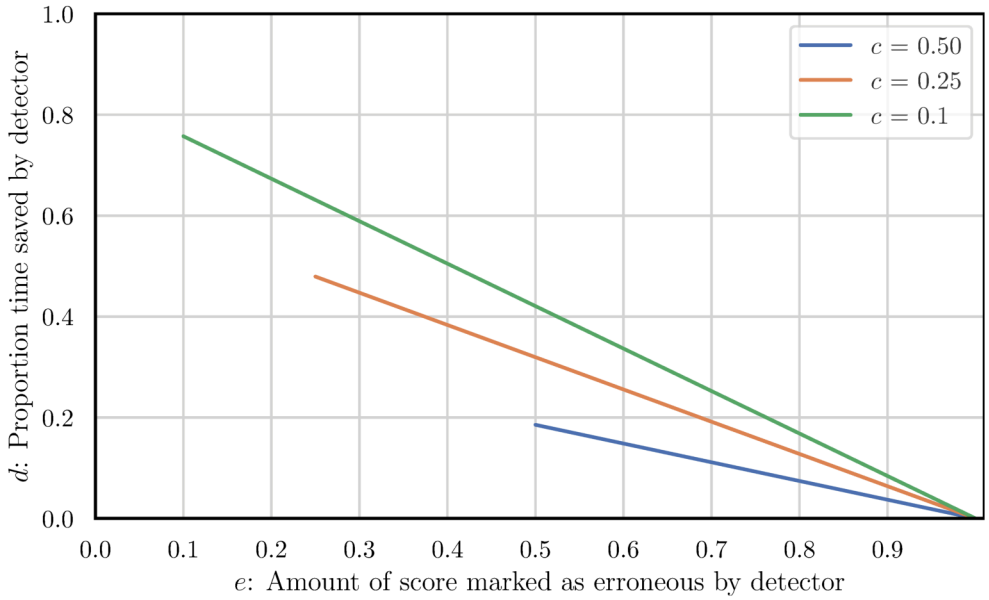
These established bounds $0.24 \leq p \leq 0.59$ suggest that in the process of identifying and correcting OMR-processed scores, task-finding will, on average, account for at least 24% and no more than 59% of the total task time. This range indicates the potential time savings that an error detector could provide. The higher value (0.59) is more optimistic, suggesting that if a significant portion of the correction process is spent in error identification, the error detector could substantially reduce the overall time required for correction. The lower value (0.24) is more conservative, indicating a lesser potential for time savings.

With these values for p it is now possible to use Equation 3.6 to calculate estimates for d, the estimated proportion of time saved by use of the error detector. Figure 3.13 shows estimates of d showing the potential reduction in correction time for various values of e (the proportion of the score identified as erroneous by the detector). Two graphs are shown for the upper and lower estimates of p. The lines in the graphs terminate at lower values of e because the detector must mark a number of errors at least equal to the actual number present in the score; $e \geq c$ must hold for the assumption of near-perfect recall to be valid. From these graphs, it is clear that the error detector's ability to reduce correction time is limited for OMR outputs with a high number of errors, as most of the score would need to be checked regardless. However, in cases where the OMR output contains few errors, a significant portion of the correction process involves searching for these errors. Here, the error detector has the potential to halve the correction time or better, particularly if the model's precision is high. These insights will be applied in

Chapter 6 to estimate the time savings for each piece in the test set when corrected with the aid of the error detector. Though based on a simplified model of the OMR correction process, this approach offers a practical way to estimate the detector's effectiveness in streamlining the OMR correction process.



(a) The graph with $p = 0.24$, simulating the case where searching for errors takes less time.



(b) The graph with $p = 0.59$, simulating the case where searching for errors takes more time.

Figure 3.13: Two graphs of Equation 3.6, which estimates d, how much an error detector could reduce OMR correction times. Each shows the effect of varying e, the proportion of the score marked as erroneous by the error detector, for three different of c, the proportion of the score that requires correction. The two graphs each use different values of p. The graphs cut off to ensure that e is always greater than c in the formula.

Chapter 4

Classification and Machine Learning

In this chapter, I focus on classification and machine-learning techniques within the context of error detection in symbolic music, to provide the necessary technical foundations for describing my proposed error detection method in Chapter 5.

I begin by defining the fundamental concept of binary classification, which involves categorizing individual elements of a dataset into one of two classes. I will discuss the challenges associated with this task and examine different performance metrics for evaluating and optimizing binary classification algorithms. Additionally, I will describe the unique methodologies used in the field of high-recall information retrieval. As discussed in Section 1.3.3, error detection is in a class of problems which aims to eliminate false negatives in classification altogether, so as to minimize the amount of time that a human must review the classifier's output.

Moving forward, I will examine common machine-learning architectures that have proven effective in error detection tasks similar to symbolic music error detection, including feed-forward neural networks and recurrent neural networks, and the backpropagation and stochastic gradient descent methods used to train them.

Attention mechanisms, which play a vital role in modern machine-learning models and serve as the central mechanism of the error detector developed in Chapter 6, will be thoroughly described. I cover the basics of attention models and their extensions in Transformer models utilizing multi-head self-attention. I address the computational challenges associated with attention mechanisms, including the problem of their quadratic time and space complexity, and the many methods that have been developed in recent years to improve their computational efficiency when operating on long sequences.

Lastly, I will discuss the relationship between machine learning and long-term dependencies in

symbolic music. As discussed in Chapter 3, many music compositions exhibit intricate patterns and relationships spanning extended durations, but it is widely acknowledged that most machine-learning methods struggle to effectively learn from these long-term dependencies. I will present various efforts made to quantify this failure and highlight existing research on identifying the underlying causes.

I will use the following conventions in the equations in this section:

- *Variables* that are inputs or outputs to a function will be italicized; e.g., $f(x) = y$
- *Constants* and *hyperparameters* of neural networks will be lowercase Greek letters: e.g., α, β, η.
- Trainable *weights* of a function, such as the parameters of a neural network, will be in boldface; e.g., a vector of biases $\mathbf{b}$.
- Wherever a variable, constant, or parameter represents a two-dimensional *matrix*, it will be written using uppercase letters; e.g., a matrix of weights that contains parameters of a neural network is written $\mathbf{W}$.
- *Datasets* containing a number of data points will be written in uppercase calligraphic font; e.g., $\mathcal{D}, \mathcal{X}, \mathcal{Y}$. A matching lowercase letter with one or more subscripts refers to an *element* of the dataset; e.g., $d_i \in \mathcal{D}$ refers to the ith item of $\mathcal{D}$.
- *Functions* and *evaluation metrics* may be written in a variety of styles according to the most commonly used conventions in machine-learning literature, so as to match the references used; e.g., a kernel function ϕ, a loss function L, a generic single-variable function f.

4.1 Binary Classification Problems

Error detection is a *classification* task. Given a number of tokens (words, musical symbols, pixels in an image, etc.), classification tasks assign a single *label* to each one. In the case where there are only two possible labels, the task is a *binary classification* problem, and the two classes are referred to as positive (an error, a member of the class of interest) or negative (not an error, not a member of the class of interest). In the case of more than two possible classes, such as where one wants to note the type of error from a given taxonomy of errors, the task is

multiclass classification. This chapter concerns only *supervised classification* tasks; the qualifier "supervised" is used when one has a dataset of examples along with their associated correct labels from which to create a classification algorithm (Kotsiantis 2007). The *semi-supervised* and *unsupervised* cases, where only some or none of the examples have associated labels, will not be covered in this chapter.

I use a review of binary classification metrics and common pitfalls by Powers (2020) as a primary source for this section. Canbek et al. (2017) compiled an exhaustive list of binary classification evaluation methods as seen in the literature, and arranged them into a hierarchical taxonomy, which I use to organize this section.

4.1.1 Accuracy Metrics for Binary Classification Problems

Supervised binary classification problems involve:

- A dataset $\mathcal{X}$ containing n feature vectors $x_0 \dots x_n \in \mathbb{R}^d$.
- A set of corresponding labels $\mathcal{Y}$ containing n labels $y_0 \dots y_n \in \{0, 1\}$, corresponding to the feature vectors in $\mathcal{X}$.
- A set of corresponding predicted labels $\hat{\mathcal{Y}}$ containing n labels $\hat{y}_0 \dots \hat{y}_n \in \{0, 1\}$, or in $\mathbb{R}$.

The *base measures* of performance for a binary classification problem count where $\mathcal{Y}$ and $\hat{\mathcal{Y}}$ agree on labels for both positive and negative examples.

- The *True Positive* measure (TPR) is the number of positive examples correctly predicted.
- The *False Positive* measure (FPR), also called the *Type I Error*, is the proportion of negative examples incorrectly predicted as positive, equal to $n - TP$.
- The *False Negative* measure (FNR), also called the *Type II Error*, is the number of positive examples incorrectly predicted as negative.
- The *True Negative* measure (TNR) is the number of negative examples correctly predicted as negative, equal to $n - FN$.

Each of these measures should be understood as functions that operate on a set of labels and a set of predicted labels. As is the convention in literature, I will refer to them using only

their abbreviations, with the understanding that (for example) TP denotes $TP(\mathcal{Y}, \hat{\mathcal{Y}})$ wherever it appears.

The term *accuracy* is casually used to refer to the performance of a classification system in general, but it has a specific meaning that can be misleading:

$$\text{Acc}(\mathcal{Y}, \hat{\mathcal{Y}}) = \frac{TP + TN}{n} \tag{4.1}$$

Canbek et al. (2017) call this "perhaps the most abused metric in binary classification performance reports." Any significant *class imbalance* in the labels (i.e., positive labels being more or less likely than negative ones) can result in an algorithm achieving a high accuracy score by predicting the same label for all data points.

Precision and *recall* are combinations of base measures that are widely used, defined in Equation 4.2.

$$\text{Precision}(\mathcal{Y}, \hat{\mathcal{Y}}) = \frac{TP}{TP + FP}, \quad \text{Recall}(\mathcal{Y}, \hat{\mathcal{Y}}) = \frac{TP}{TP + FN} \tag{4.2}$$

Precision represents the proportion of items predicted as positive that actually possess positive labels, while recall measures the proportion of items with positive labels that were correctly predicted as positive. These metrics are inherently in conflict with each other, as highlighted by Buckland and Gey (1994); improving one metric necessarily results in a decrease in the other. Consequently, directly optimizing both precision and recall simultaneously is not feasible in practice.

Various measures have been developed to assess the overall performance of a binary classifier using a single metric. One widely utilized measure is the F_β-Measure, which calculates a weighted harmonic mean of precision and recall:

$$\text{F}_\beta(\mathcal{Y}, \hat{\mathcal{Y}}) = (1 + \beta^2)\frac{Pr \cdot R}{Pr \cdot \beta^2 + R} = \frac{(1 + \beta^2)TP}{(1 + \beta^2)TP + \beta^2 \cdot FP + FN}. \tag{4.3}$$

In the context of the F_β-Measure, a higher value of the parameter β assigns greater importance to recall compared to precision, while a lower value does the opposite. The special case when $\beta = 1$ is known as the F_1-measure, or simply the F-measure. Though widely used, this measure has certain limitations, as highlighted by Powers (2020). One of its drawbacks is that it is insensitive to the number of samples correctly identified as negative (TN), which is undesirable when dealing with imbalanced datasets or when prioritizing high precision. This

can artificially inflate the performance of classifiers that heavily favor positive predictions and can lead researchers to overestimate the true effectiveness of a classifier.

A more robust measure is the *Matthews Correlation Coefficient* Matthews Correlation Coefficient (MCC):

$$\mathrm{MCC}(\mathcal{Y}, \hat{\mathcal{Y}}) = \frac{TP \cdot TN - FN \cdot \ FP}{\sqrt{(TP + FP)(TP + FN)(TN + FN)(TN + FP)}}. \tag{4.4}$$

The MCC produces a value ranging between -1 and 1, where -1 indicates perfect misclassification, 1 represents perfect classification, and 0 is the expected value for a random predictor. Chicco and Jurman (2020) discusses several advantages of the MCC over traditionally used scores such as Accuracy and F_β-Measures. One significant advantage is that the MCC considers both positive and negative examples and rewards a high score only when both classes are consistently evaluated accurately. Additionally, the MCC automatically accounts for class imbalance without requiring users to specify any additional parameters.

4.1.2 Thresholding

In most cases, a classification algorithm does not directly output labels but rather assigns a real-valued score to each item, which must be converted into binary predictions before evaluation. This requires a threshold to be selected to divide the classifier's scores into positive and negative predictions. When the true labels are available, one can experiment with various thresholds to maximize some chosen performance metric (e.g., the F_β score). However, when using a classifier for inference, it becomes necessary to make an educated guess regarding the optimal threshold. A lower threshold will increase recall but decrease precision, while a higher threshold will have the opposite effect; depending on the ultimate goal of the classifier, this trade-off must be taken into consideration.

An intuitive approach is to empirically determine the optimal threshold using a dataset where the labels are known and then apply this threshold to all unseen data during inference. However, Lipton et al. (2014) demonstrate that choosing a threshold based on the F_β score on a limited dataset may lead to suboptimal future performance, as the F_β score treats positive and negative labels unequally. A comparison of threshold-setting strategies by Freeman and Moisen (2008) found that selecting a threshold that maximizes the *kappa criteria*, which is equivalent to the MCC for binary classification tasks, resulted in consistently acceptable performance on

unseen data.

Some metrics for evaluating classifier performance avoid the need to choose a specific threshold by aggregating statistics across all possible thresholds. The *Receiver Operating Characteristic* Curve (ROC Curve) of a classifier plots the true positive rate (equivalent to recall) against the true negative rate (equivalent to $\frac{TN}{TN+FP}$) for all possible thresholds on a particular dataset. A random classifier's ROC curve forms a diagonal line, while a useful classifier will exhibit an upward-arching curve away from the central diagonal. The Area Under the ROC Curve (AUC) is a quantitative measure summarizing the classifier's performance across all possible thresholds (Hanley and McNeil 1982). Similarly, an AUC-like metric can be derived by calculating the area under the precision-recall curve for various thresholds (Sofaer et al. 2019). These metrics are useful for comparing the performance of different classifiers, but they do not eliminate the need to select a threshold during inference when encountering unknown data.

4.1.3 High-Recall Information Retrieval

High-Recall Information Retrieval (HRIR) pertains to binary classification tasks where achieving a recall score of 1, or close to 1, is crucial. These types of applications are found in domains where a false positive has minimal consequences, but a false negative is undesirable. Examples of such domains include *electronic discovery*, which involves retrieving all relevant documents based on a user's query, necessary in tasks like patent document retrieval (Shalaby and Zadrozny 2019). High-Recall Information Retrieval (HRIR) tasks also arise in error detection, such as in identifying defects in automotive products, where overlooking a single defective item can have catastrophic implications (Li et al. 2021). In most HRIR scenarios, it is assumed that human intervention is required to review the algorithm's positive classifications and eliminate false positives. The objective of HRIR is to minimize the number of false positives that necessitate manual evaluation by the user, while ensuring that the algorithm does not produce false negatives by incorrectly excluding items.

Classifiers employed in HRIR adopt a variety of techniques to enhance precision that are usually tailored to the specific domain they operate in. As an example in the domain of error detection, Ambati et al. (2010) developed a classifier for error detection in Hindi sentences by first identifying infrequent phenomena and subsequently utilizing rule-based algorithms that incorporate known rules of syntax and grammar to reduce this set of uncommon occurrences. This strategy initially casts a wide net, resulting in high recall but low precision, and then

employs a rule-based post-processing step to improve precision to some extent.

Conventional performance metrics for binary classification systems are not well-suited for HRIR. While the F_β-measure (Equation 4.3) can be utilized with a high value of β to prioritize recall over precision, it still does not consider true negatives. As a result, it is unable to distinguish between an HRIR classifier that correctly excludes a thousand data points from requiring human review and one that excludes only ten. The Matthews Correlation Coefficient (MCC) (Equation 4.4) performs better in this context, but it assigns equal weight to the classifier's performance on positive and negative items, which does not align with the goals of HRIR. In many high-recall applications, the ultimate goal is to minimize the human effort involved in the process, but running user trials to quantify this reduced effort is costly (Zhang et al. 2018).

In the context of HRIR, performance metrics typically operate on a full *ranking* of dataset elements rather than pre-thresholded predictions. In this ranking, each item in the dataset $\mathcal{X}$, indexed by i, is assigned a unique natural number in ascending order: r_i denotes the ranking of element $x_i \in \mathcal{X}$. Lower numbers correspond to higher ranks and, consequently, higher scores from the classifier. To simplify notation, the ranking is alternatively expressed as the *cumulative ranking* $R_0 \dots R_i$, where R_j represents the number of positively labeled items with scores higher than the item at rank r_j. In other words, R_j signifies the count of positively labeled items that would be found by considering only the j items ranked highest by the classifier (Note that the normal definition of the ranking counts all items, not just those that are positively labeled). Working with rankings eliminates the need to select a threshold for classifier output and allows for the consideration of system performance aspects relevant to human interaction. Rankings enable the researcher to address questions such as: "On average, how many of the top-ranked items must a human review to have 99% confidence of not missing any positive data points?"

The most commonly used ranking-based metric in HRIR is the *average precision* (AP), defined as:

$$\mathrm{AP} = \frac{1}{p} \sum_{i=1}^{p} \frac{R_i}{i}, \tag{4.5}$$

where p is defined as the number of items in $\mathcal{X}$ with positive labels (Robertson 2008). A value of 1 is achieved when all p positively labeled items in the dataset are the p highest-ranked ones, resulting in $R_i = i$ for all i. The motivation behind this metric is to implicitly simulate various human retrieval strategies simultaneously. The summation terms correspond to the

number of negatively labeled items that would need to be examined to find a certain number of positively labeled items. This models the behavior of users with varying degrees of tolerance for false positives. However, Moffat and Zobel (2008) criticized AP for modeling unrealistic "estimates of user satisfaction," arguing that the human retrieval strategies that AP simulates do not constitute a coherent, interpretable metric reflecting the overall usability of the system.

An alternative metric to AP is the *normalized recall* (R_{norm}), which was initially defined by Rocchio (1964) as the "normalized recall index" and adapted in this chapter (Equation 4.6) with suitable algebraic and notational changes to match the conventions of this chapter.

$$R_{norm} = 1 - \frac{1}{p(|\mathcal{X}| - p)} \sum_{i=1}^{p} (r_i - i) \tag{4.6}$$

R_{norm} compares a given ranking to the best and worst possible rankings, as represented by the difference inside the summation. The remainder of the formula then normalizes this comparison to the range of [0, 1]. Evaluating R_{norm} incorporates information about the dataset size and requires a ranking for every single element, which can be impractical for large collections. To address this, Magdy and Jones (2010) proposed the "Patent Retrieval Evaluation Score" (PRES), which introduces a parameter N representing the maximum number of items a human would be willing to review. The PRES metric is defined in (Equation 4.7).

$$\begin{aligned} \text{PRES} &= 1 - \frac{1}{N} \left(\frac{S}{p} - \frac{p+1}{2} \right), \\ \text{where } S &= \sum_{i=1}^{N} (r_i - i) i + R_N (N + p) \end{aligned} \tag{4.7}$$

Here an item that is given a high score by the classifier has a low numerical rank, so the intermediate term S is small if most positively labeled items are confidently marked as positive by the classifier. The parameter N serves as a proxy for the total dataset size.

4.2 Machine-Learning Architectures and Training Techniques

In this section, I provide an introduction to common machine-learning architectures and training techniques within the context of binary classification problems. The discussed architectures include Feed-forward Neural Network (FNN)s, Recurrent Neural Network (RNN)s, and Long Short-Term Memory (LSTM) networks. Additionally, I define the concepts of stochastic gradient

descent and backpropagation, which are fundamental to training these architectures effectively.

4.2.1 Feed-Forward Neural Networks

The Feed-forward Neural Network (FNN), also known as a *multi-layer perceptron*, consists of individual *fully connected layers*, or *perceptrons*. Each fully connected layer is a function that takes an input vector $x \in \mathbb{R}^n$ and produces an output vector $y \in \mathbb{R}^m$. The layer is defined by a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times m}$, a bias vector $\mathbf{b} \in \mathbb{R}^m$, and a nonlinear activation function ϕ applied element-wise to the input. The general formulation of a single feed-forward layer is given by:

$$y = \phi(x\mathbf{W} + \mathbf{b}). \tag{4.8}$$

A three-layer FNN, with different sets of weights and biases for each layer, can be represented as:

$$y = \phi(\phi(\phi(x\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2)\mathbf{W}_3 + \mathbf{b}_3). \tag{4.9}$$

The number of layers in an FNN is referred to as the *depth* of the network. Each layer can have an output vector of varying size, provided that the weight matrices and biases are appropriately sized for valid matrix operations. The width of a particular layer is defined as the size of its output vector. The width of the final layer determines the shape of the network's output y. Figure 4.1 shows a schematic of a network of depth 3.

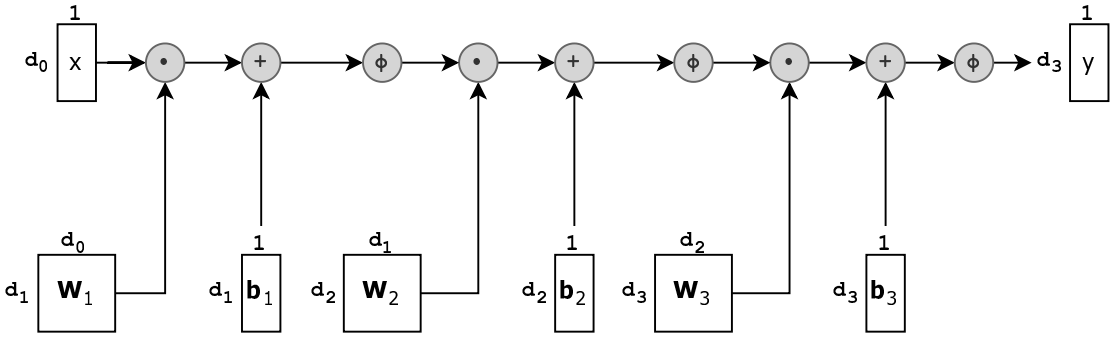


Figure 4.1: A schematic of the matrix operations involved in a three-layer FNN.

To be useful, the activation function of an FNN must be nonlinear, as otherwise the weights and biases would commute with each other through the activation function, reducing the entire FNN to a single affine transformation. In earlier years of neural network research, the sigmoid and hyperbolic tangent functions were commonly used as activation functions. The sigmoid

function is defined as

$$\text{sig}(x) = \frac{1}{1 + e^{-z}}, \tag{4.10}$$

while the hyperbolic tangent function is defined as

$$\tanh(x) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{4.11}$$

These functions were popular due to their differentiability and their resemblance to the step functions used in earlier single-layer perceptron models in the history of machine-learning research (Briot et al. 2019, p. 49).

Hornik et al. (1989) demonstrated that a FNN utilizing the sigmoid activation function, and consisting of at least two layers with arbitrarily large width, has the ability to approximate any continuous function from the input space to the output space, a property known as *universal function approximation*. It implies that for every continuous function, there exist weight matrices $\mathbf{W}_1, \mathbf{W}_2$ and bias vectors $\mathbf{b}_1, \mathbf{b}_2$ for each of the two layers that enable the FNN to approximate the function with arbitrary precision. Leshno et al. (1993) extended the findings of Hornik et al. by demonstrating that universal function approximation holds true for any choice of activation function ϕ used in the network, with the exception of polynomial functions.

In recent years, the rectified linear unit (ReLU) has gained significant popularity and has become the standard activation function used both in FNNs and other machine-learning architectures. The ReLU function is defined as

$$\text{ReLU}(x) = \max(0,\ x). \tag{4.12}$$

Its simplicity, ease of implementation, and the lack of compelling evidence showing performance improvements by adopting more complex activation functions have contributed to its widespread use (Briot et al. 2019, 48).

To introduce FNNs into the context of binary classification, I define:

- $f_{\mathbf{p}} : \mathbb{R}^d \to \mathbb{R}$ to be a function comprising an FNN with parameters (weight matrices and bias vectors) represented by the single character $\mathbf{p}$ and a final layer of width 1,
- $\mathcal{X}$ to be a set of n datapoints $x_i \in \mathbb{R}^d$, each of which is a vector that could be input to f,

- $\mathcal{Y}$ to be a set of n binary classification labels $y_i \in \{0, 1\}$, where each y_i corresponds to the desired classification of the data point x_i.

From this, I can define a *loss function* L that quantifies the accuracy of the FNN on a given dataset given a set of parameters $\mathbf{p}$:

$$L(\mathbf{p}) = \frac{1}{n} \sum_{i=0}^{n} |f_{\mathbf{p}}(x_i) - y_i| \tag{4.13}$$

This formulation, known as the *mean absolute error loss*, represents one of many possible loss functions. The particular loss function chosen will imply different characteristics about the optimal model. For example, the *mean squared error* loss, shown in Equation 4.14 penalizes highly incorrect predictions more than the mean absolute error loss does, because it replaces the absolute value operation with a square.

$$L(\mathbf{p}) = \frac{1}{n} \sum_{i=0}^{n} (f_{\mathbf{p}}(x_i) - y_i)^2 \tag{4.14}$$

The *binary cross-entropy loss*, shown in 4.15, often used in binary classification problems, is suitable when the labels y_i are always members of the set $[0, 1]$, as in binary classification problems. For each datapoint, the first term of the summation vanishes when the target label is 0 and the second term vanishes when the label is 1. The ideal model defined by this loss function is one that outputs, for each datapoint, a prediction in the form of the probability that the point has a positive label.

$$L(\mathbf{p}) = -\frac{1}{n} \sum_{i=0}^{n} y_i \log f_{\mathbf{p}}(x_i) + (1 - y_i) \log(1 - f_{\mathbf{p}}(x_i)) \tag{4.15}$$

The problem of training a neural network to perform binary classification is thus expressed as the problem of finding a set of parameters $\mathbf{p}$ that minimizes $L(\mathbf{p})$ for given sets of datapoints $\mathcal{X}$ and target classifications $\mathcal{Y}$.

4.2.2 Gradient Descent

In the case of feed-forward neural networks (FNNs) and other networks derived from them, the process of *gradient descent* is employed to find optimal sets of weights and biases. Gradient descent is an iterative optimization technique that is particularly useful for finding minimal or maximal values of functions that are challenging to compute analytically.

Gradient descent begins with an initial guess for the set of parameters $\mathbf{p}$, denoted as $\mathbf{p}_0$. The first step is to compute the *gradient* of the function f at $\mathbf{p}_0$, denoted as $\nabla f(\mathbf{p}_0)$. The gradient is a vector obtained by taking the derivative of f with respect to each parameter $\mathbf{p}$, and it points in the direction where f increases most rapidly in the parameter space starting from the initial point $\mathbf{p}_0$. Based on this initial guess, gradient descent updates its estimation of the optimal $\mathbf{p}$ using the update scheme shown in Equation 4.16.

$$\mathbf{p}_{n+1} = \mathbf{p}_n - \eta \nabla f(\mathbf{p}_n) \tag{4.16}$$

Here, η represents the *learning rate*, a tunable parameter of the gradient descent algorithm. The learning rate controls the step size that the algorithm takes in each iteration. The algorithm continues to run until the value of $f(\mathbf{p}_n)$ no longer decreases, at which point $\mathbf{p}_n$ is considered the optimal value. However, this process does not guarantee arriving at a global minimum of the function, and different choices for $\mathbf{p}_0$ and η can lead to convergence on different local minima. Various adaptations and variations of this formulation exist to enhance the robustness of the optimization process. A commonly used enhancement to gradient descent is the inclusion of *momentum*:

$$\mathbf{p}_{n+1} = \mathbf{p}_n - \eta \nabla f(\mathbf{p}_n) + \alpha(\mathbf{p}_{n-1} - \mathbf{p}_n) \tag{4.17}$$

The additional term involving the parameter α introduces momentum to the gradient descent process. It incorporates a fraction of the gradient from the previous step into the current update. This momentum allows the algorithm to "push past" insignificant local minima if the overall trend of the gradient updates indicates a particular direction. Choosing appropriate values for the learning rate η and the momentum α is crucial; if the values are too small, the convergence may be unacceptably slow or restricted to local minima. Conversely, large values can cause the algorithm to diverge and fail. In some cases, the performance can be improved by utilizing a learning rate *scheduler*, which dynamically adjusts η and α as the algorithm progresses. A common approach is to reduce the learning rate by a constant factor after a certain number of updates (Robbins and Monro 1951).

4.2.3 Backpropagation and Stochastic Gradient Descent

Performing gradient descent for a loss function defined using an FNN requires some method of taking the gradient of an FNN with respect to its parameters $\mathbf{p}$, for an arbitrary dataset $\mathcal{X}$ and labels $\mathcal{Y}$. The technique of *backpropagation* allows computation of the gradient of the loss function for any feedforward architecture.

At a high level, backpropagation consists of a *forward pass* and a *backward pass*. During the forward pass, each data point from $\mathcal{X}$ is fed through the network, producing a set of predicted data points $\hat{\mathcal{Y}}$. These predictions are then compared with the corresponding labels from $\mathcal{Y}$ to evaluate the network using a loss function, such as the binary cross-entropy loss defined in Equation 4.15. Throughout this process, the algorithm computes the partial derivatives between the inputs and outputs of each operation defined within the network (such as matrix multiplications, activation functions, and their compositions), storing these derivatives in memory.

In the backward pass, the algorithm starts from the final layer and works its way backward to the input layer. It combines the stored derivatives using the chain rule to calculate the gradients layer by layer. At the end of this process, the derivatives remaining at the input layer form the gradient of the loss function. This gradient can then be utilized in a gradient descent algorithm (Equation 4.16) to optimize the weights of the network.

I present this high-level explanation to emphasize two crucial aspects. First, backpropagation relies on the differentiability of all components of the network, since each part must be individually differentiated. Second, it necessitates the storage of intermediate derivative values in memory for every operation performed by the neural network. This can become a performance bottleneck, especially for networks handling large inputs, as discussed in Section 4.3.4. For a comprehensive treatment that derives backpropagation from fundamental calculus principles and provides proofs of its applicability to various neural network architectures, I recommend referring to Rojas's introduction to the backpropagation algorithm (1996, p. 153-162).

In practice, evaluating $\nabla L(\mathbf{p})$ using the entire dataset $\mathcal{X}$ at every step of the gradient descent algorithm is often infeasible, especially for large datasets. Instead, a subset of $\mathcal{X}$ is used to approximate the gradient in each step, with this subset changing for every gradient update. This modification is known as Stochiastic Gradient Descent (SGD). Each subset of $\mathcal{X}$ used for a gradient update is referred to as a *batch*, and a series of gradient updates using a collection of batches that cover the entire dataset is called an *epoch*. Since only a fraction of the dataset is utilized in each gradient update, the approximated gradient of the loss function can exhibit

significant fluctuations between updates. This added noise may lead to less reliable convergence of the network. However, networks trained with small batch sizes tend to generalize better on unseen data compared to models trained with large batch sizes (Li et al. 2014). Similar to the learning rate, determining the optimal batch size for a specific training configuration can significantly impact the training behavior and the ultimate performance of a network. For comprehensive benchmarks exploring the effects of these two hyperparameters on the performance of FNNs in simple classification problems, Breuel (2015) provides a valuable resource.

In the case of sufficiently deep neural networks, employing a single learning rate across all trainable parameters and throughout all learning epochs may not yield satisfactory convergence. Learning rate schedulers can be employed, but the types of simple schedulers that succeed in simpler gradient-descent problems may not be suitable for deep neural networks. The loss functions of complex networks can be less well-behaved, and their SGD updates often display more chaotic behavior (Ge et al. 2019). Recent research has shown the effectiveness of *cyclical learning rate* scheduling for SGD, where the learning rate undergoes repeated increases and decreases during training. While this approach occasionally results in performance dips, it allows the model to escape from local minima more easily (Smith 2017). By employing cyclical learning rate scheduling that occasionally reaches an extremely high maximum learning rate, typically considered unworkable in theoretical gradient descent research, a phenomenon known as *super-convergence* can be induced, wherein an SGD algorithm achieves convergence in a significantly smaller number of epochs than usual (Smith and Topin 2018).

A breadth of literature focuses on optimization algorithms for SGD, commonly referred to as *optimizers* in the context of machine learning. These optimizers dynamically adjust learning rates and momentum values for individual weight parameters during the course of training. A common approach involves updating components of the network that infrequently receive updates with a higher learning rate, while lowering the learning rate for components that are frequently activated. Ruder (2017) provided an review of these strategies along with a comprehensive examination of commonly used optimizers. Among them, the Adam optimizer introduced by Kingma and Ba (2017) is identified as a suitable choice for most applications; however, they noted that the performance differences between recent optimizers are small for most problems.

4.2.4 Recurrent Neural Networks

While FNNs theoretically have the capability to handle sequential data by concatenating sequences into long vectors, this approach is impractical for long sequences and requires fixed-size input and output. To address these limitations, the Recurrent Neural Network (RNN) was developed specifically for sequential data. RNNs can handle sequence classification tasks, where a single label is assigned to the entire input sequence, as well as sequence-to-sequence mapping tasks, where one sequence is transformed into another. RNNs have no restrictions on the length of their input and output and a single network can process sequences of any length (Briot et al. 2019, 78).

The structure of the simplest RNN resembles that of an FNN, with fixed-size input and output and one or more *recurrent layers*. In an RNN, items from an input sequence are sequentially fed into the network, one at a time. Each recurrent layer generates an output, which is stored in memory and appended to the next input. This ensures that every layer has access to the current sequence element as well as its own hidden state from the previous time step. Typically, the hidden state for the first time step is initialized as a vector of zeroes. For a sequence $x_0, x_1, x_2, ..., x_n$ fed into a single recurrent layer L, the outputs $y_0, y_1, y_2, ..., y_n$ at each time step can be represented as:

$$\begin{aligned} y_0 &= \mathbf{W}(x_0 \parallel \vec{0}) \\ y_1 &= \mathbf{W}(x_1 \parallel y_0) \\ &\vdots \\ y_n &= \mathbf{W}(x_n \parallel y_{n-1}) \end{aligned} \tag{4.18}$$

Here, $\vec{0}$ represents a zero vector, and $\parallel$ denotes the vector concatenation operation. Figure 4.2 shows a schematic of this simple architecture.

RNNs are trained in a supervised manner by comparing the output at each time step with the corresponding items in a target sequence. In the common case of predicting the next element in a sequence (predicting x_n given $x_0, x_1, ..., x_{n-1}$), the target sequence is the same as the input sequence, but shifted backward in time. The backpropagation algorithm needs to be modified to enable gradient flow along the recurrent links in the recurrent layers. The standard technique used for this purpose is called *backpropagation through time* (Werbos 1990). Conceptually, this

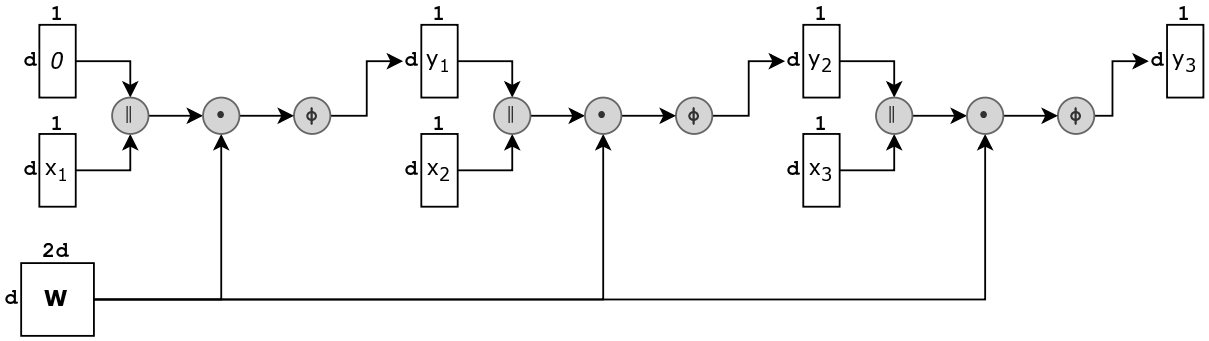


Figure 4.2: A schematic of the matrix operations involved in an RNN operating on a sequence of length 3.

technique "unrolls" the RNN, representing it as an extended graph of identical layers sharing a common set of weights and biases, as represented in Equation 4.19. In this unrolled representation, the network is non-recurrent, so backpropagation can proceed as normal. In principle, a significant loss incurred on the nth time step can propagate backward to the 0th time step, thus modifying the weights of the network to encourage the retention of specific information in the hidden state.

Despite this theoretical capability, empirical evidence has demonstrated that RNNs relying solely on passing hidden states forward in time struggle to capture dependencies in their data. An influential experiment by Mozer (1991) highlighted this issue by using a dataset of strings in the form `xSx`, where `x` is a randomly chosen character and `S` is a fixed sequence shared by all strings. The task was to predict the next character given an incomplete string. Surprisingly, the recurrent network failed to learn the simple rule that the last character is always the same as the first, even when `S` was as short as four characters; by the fourth time step, the network had effectively "forgotten" the initial information it had seen.

The vanishing gradient problem, a well-documented issue in RNNs, explains this behavior (Bengio et al. 1994). The problem arises from the repeated multiplication of the same weight matrix into the hidden state at later time steps. Analyzed by Hochreiter (1998), backpropagation through time involves multiplying by a series of terms whose absolute values can either be greater or less than one, depending on the eigenvalues of the weight matrix $\mathbf{W}$. This multiplication can cause gradients to either vanish or explode, hindering the flow of error signals and impeding learning in RNNs. The impact of repeated matrix multiplication becomes evident in the unrolled expression for the third output y_2 of an RNN:

$$y_2 = \mathbf{W}(x_2 \parallel \mathbf{W}(x_1 \parallel \mathbf{W}(x_0 \parallel \vec{0}))) \tag{4.19}$$

When the eigenvalues of $\mathbf{W}$ are less than one, the gradient decays exponentially as it propagates further back in time, resulting in the dominance of recent time steps in the gradient. Using larger learning rates does not alleviate this issue, as the ratio between the effects of recent and distant time steps remains the same. Conversely, if the eigenvalues of $\mathbf{W}$ are greater than one, the hidden state tends to grow exponentially or oscillate excessively, and the network fails to train.

The choice of activation function, particularly the use of sigmoid activation functions and similar "S"-shaped curves (e.g., the hyperbolic tangent), can contribute to the vanishing gradient problem. The gradients of these functions are significantly larger than zero only within a small interval and are less than 1 everywhere else. Consequently, gradients can vanish early in training due to the diminishing effect of repeatedly passing through these activation functions. Vanishing gradients are part of a broader class of phenomena present in all gradient descent systems, indicating that training can fail in RNNs regardless of the specific activation function employed (Hochreiter et al. 2001).

4.2.5 Long Short-Term Memory Networks

The LSTM network was introduced by Hochreiter and Schmidhuber (1997a) to address the challenges of vanishing and exploding gradients in RNNs. Since LSTM networks are a form of recurrent network, for clarity, I use the term RNN specifically to refer to the simple architecture described in the previous section.

The key insight behind the design of LSTMs is the recognition that weights in recurrent layers of an RNN need to manage the storage and forgetting of information over multiple time steps. This dual role often leads to a phenomenon of *conflicting weight updates*, described formally by Staudemeyer and Morris (2019), wherein gradient updates from different timesteps contradict and cancel each other out. LSTMs address this issue by explicitly assigning separate components within the network to handle each conflicting function, operating in parallel. LSTMs enhance the recurrent layers of RNNs with *memory cells*. In addition to the cell's output, which is passed to the next component of the network and concatenated with the input of the next time step, similar to a standard RNN, each memory cell maintains an internal state known as the *cell state*.

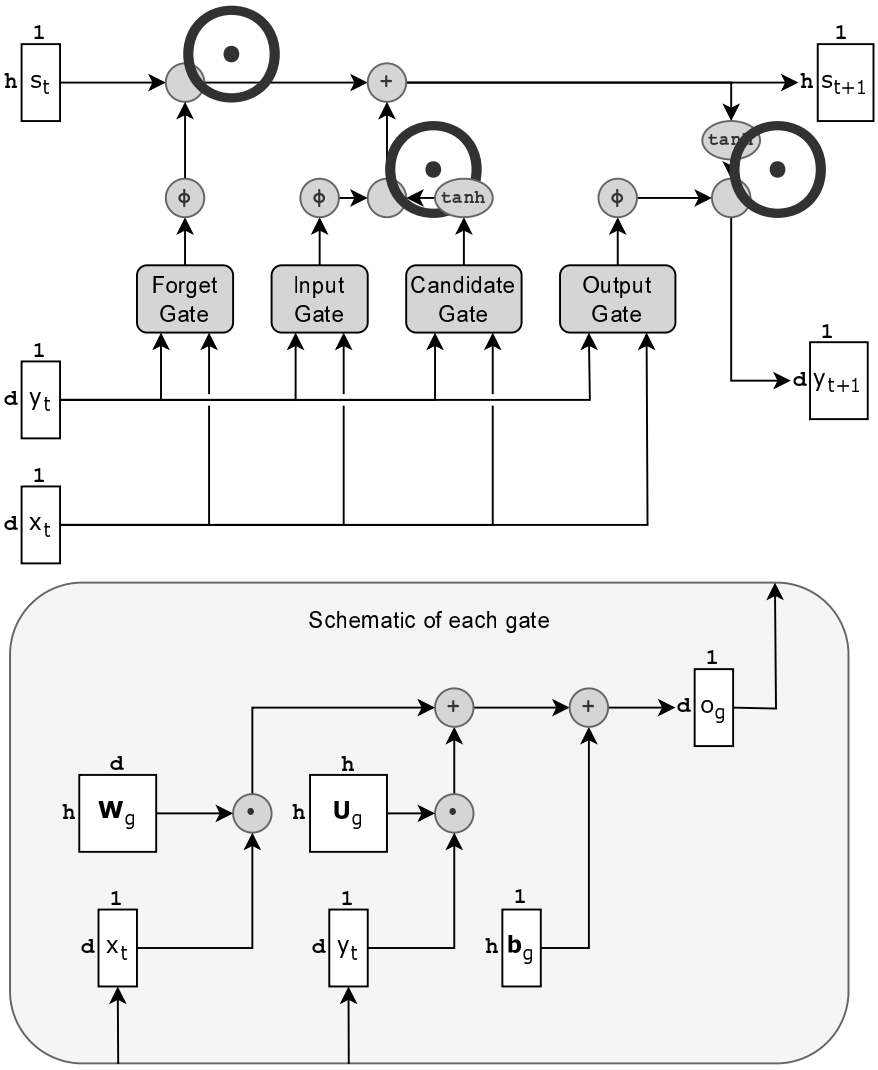


Figure 4.3: A schematic of the matrix operations involved in a single LSTM memory cell. Note that each of the four gates are trained with a unique set of weights and biases. The $\odot$ symbol represents Hadamard multiplication (elementwise multiplication of vector elements).

The modification of the cell state is controlled by four *gates*, each represented by trainable neural networks. These gates take two vectors as input: x_t,the current network input, and y_{t-1}, the output from the previous time step. In the order that the gates execute on each time step of an LSTM's operation:

- The *forget gate* takes in the input of the network at that time step, and passes it through a feed-forward layer with a sigmoid activation function. This vector is then multiplied element-wise into the cell state. By doing so, the network can learn to selectively "forget" specific information within the cell state based on the input received.

- The *input gate* processes the input to the network at each time-step with a standard feed-forward layer, passes the result through a sigmoid activation function, and combined its

result with the next gate.

- The *candidate gate* is another feed-forward layer, but its output is passed through a hyperbolic tangent function, so its values lie in the range $(-1, 1)$ instead of $(0, 1)$. The outputs of the candidate gate and the input gate are element-wise multiplied, and the result is element-wise added to the cell state. Intuitively, the input gate learns to filter out irrelevant "candidates" produced by the candidate gate so that they are not added to the cell state. This allows the network to learn when to retain the current cell state and when to update it with new information from recent inputs.

- The *output gate* generates a vector that passes through a sigmoid activation function. This vector is then multiplied element-wise with the cell state. The result of this multiplication constitutes the output of the memory cell for the current time step. This enables the network to learn when specific aspects of the hidden state should be propagated to the next time step and when they should be discarded.

Memory cells composed of these components form the most common formulation of the LSTM, which I will refer to as the "vanilla" LSTM from now on. In the vanilla LSTM, the cell state is modified solely through the operation of the gates, without explicit multiplication by any trained weights. This design allows error signals to propagate backward in time along the cell state without suffering from the issues of exploding or vanishing gradients encountered in traditional RNNs. The gates in the LSTM are controlled by differentiable functions, enabling the use of backpropagation-through-time methods for gradient updates during training. Figure 4.3 shows a schematic of a single LSTM memory cell.

The most widely used variant of the vanilla LSTM is the *bi-directional LSTM*. In these networks, two layers of memory cells are employed, one processing the input sequence in the forward direction and the other processing the input in reverse. The two LSTMs run in parallel, and their outputs are concatenated at the end. This bidirectional setup enables each time step's output to access information from both past and future sequence values. Although bi-directional LSTMs are not applicable to tasks involving the prediction of future elements in a sequence, they tend to outperform unidirectional LSTMs in domains such as speech recognition and handwriting recognition (Breuel 2015).

A wide range of architectural variants have been developed for the LSTM, introducing modifications to the structure of the memory cell by adding or removing gates, altering the way the

cell state is updated at each timestep; a review of these variants can be found in Yu et al. (2019). In a comprehensive study, Greff et al. (2017) investigated the performance of numerous LSTM variants with different structural elements modified or removed across tasks such as handwriting recognition, acoustic modeling, and polyphonic music modeling. The tested variants included models that lacked or combined different gates found in the vanilla LSTM, models that replaced certain activation functions with the identity function, and the "full gate recurrence" architecture, which incorporated additional recurrent connections between all three gates. None of the architectural variants outperformed the vanilla LSTM across all three tasks. However, some variants that were strictly simpler than the vanilla LSTM, such as those that remove some gates, achieved comparable performance, suggesting that common LSTM architectures could be simplified without sacrificing effectiveness.

4.3 Attention Mechanisms

Attention, first introduced by Bahdanau et al. (2015) in the context of machine translation, is a highly flexible and modular component of neural networks that has proven to be useful for tasks involving long inputs. The fundamental idea behind attention is that a network performs a pairwise comparison between every element in two input sequences and decides the relevance of each pair based on a similarity function. This allows the network to selectively focus on specific elements in the input, internally marking them as important and controlling the flow of information from different regions of the sequence. Unlike recurrent operations, attention does not process inputs one at a time; instead, it takes the entire sequence as input simultaneously. This is in contrast to the strategy of the LSTM, which processes individual elements of an input sequence, one at a time.

An attention function takes an *input* sequence of feature vectors X with a length of n and a *state* sequence S with a length of m, and produces an *output* sequence Y, of variable length depending on the implementation. In the following equations, I treat these sequences of feature vectors as two-dimensional matrices. Each element of the matrices X, S, and Y is a real-valued vector in an embedding space of dimension b (e.g., word embeddings in Natural Language Processing (NLP)) The most basic attention mechanism, without trainable weights, can be defined as:

$$Y = \sigma(SX^\top)X \tag{4.20}$$

Here, S and $X^\top$ are matrices of size $m \times b$ and $b \times n$, respectively. The product $SX^\top$ yields an $m \times n$ matrix, where each element at index (i, j) represents the scalar dot product between X_i and S_j. This operation is commonly known as *dot-product attention*, as initially described by Luong et al. (2015).

The function $\sigma : \mathbb{R}^k \to (0, 1)^k$ is commonly known as the *softmax* function and operates individually on each row of the matrix product $SX^\top$. Its purpose is to take a real-valued vector as input and scale it to form a valid probability distribution, where the sum of its elements is equal to 1 and each element falls within the range of $[0, 1]$. The formal definition of the softmax function is given by:

$$\sigma(v) = \frac{e^{v_i}}{\sum_{j=1}^{k} e^{v_j}},\ i \in 1, ..., k \tag{4.21}$$

The exponential terms in the function equation ensure that each entry in the vector is weighted based on the magnitude of its exponential value; as a result, larger values in vector v have a much stronger influence, overpowering smaller values. This characteristic of the softmax function is reminiscent of a regular "max" function, which returns a vector with a single 1 at the location of the original vector's maximum value. However, the softmax function is fully differentiable, making it suitable for integration into machine-learning architectures. In the context of attention mechanisms, the output of the softmax function $\sigma(XS^\top)$ is referred to as the *attention weights*. Each column of the attention weights represents a distribution associated with a specific element of the sequence X. These distributions indicate which elements of S are most similar to the corresponding element of X, with the dot product serving as the measure of similarity. The final multiplication with X re-weights the sequence based on where the attention weights are strongest. The basic attention mechanism shown in Equation 4.20 contains no trainable parameters, and so is only useful as a component of another network, but it provides a foundation for understanding the key idea behind attention.

There are several variants of attention that differ in how they calculate the attention weights. *General attention*, introduced by Luong et al. (2015), is the simplest trainable variant. It includes a single trainable weight matrix $\mathbf{W} \in \mathbb{R}^{b \times b}$ between the input and state sequences, and the attention weights are computed as:

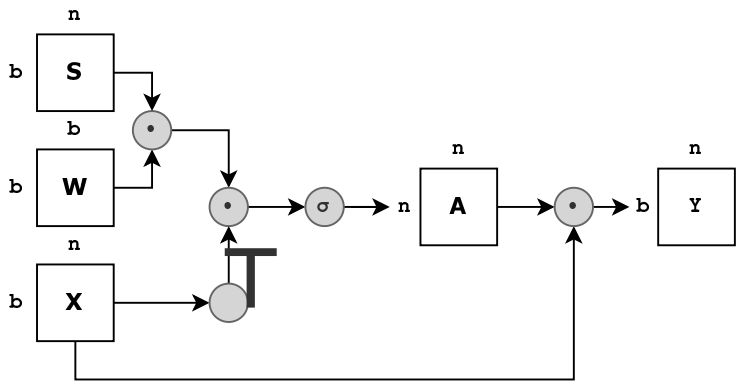


Figure 4.4: A schematic of the matrix operations involved in the general attention mechanism.

$$Y = \sigma(S\mathbf{W}X^\top)X. \tag{4.22}$$

Another variant is *additive attention*, also known as *Bahdanau attention*, which concatenates the input sequence X with the state sequence S and feeds the result through a one-layer feed-forward neural network with trainable parameters (Bahdanau et al. 2015). A schematic of general attention is shown in Figure 4.4, and a schematic of additive attention is shown in Figure 4.5.

Finally, of historical note, there is *hard attention*, first described by Xu et al. (2015). Hard attention initially calculates the attention weights using the dot-product attention formulation. However, instead of taking a weighted sum, it stochastically samples a random element from the input sequence X based on the softmax probabilities. Due to the non-differentiability of this sampling process, models using hard attention cannot be directly trained using gradient descent and backpropagation.

To summarize: the core concept of all attention mechanisms is to generate a matrix by computing a pair-wise similarity function between two sequences of vectors, followed by applying the softmax function to transform each column of the matrix into a distribution. This similarity function is typically computed using a matrix multiplication between the two sequences, but many variants exist that combine information on the two sequences in other ways. The resulting matrix, referred to as the attention weights, is then multiplied back into one of the original sequences to re-weight and emphasize particular regions of the input over others.

Note that the basic attention mechanism does not use any concept of the ordering of the elements of its input sequences. This enables attention mechanisms to diffuse information from any one sequence position to any other without needing any memory cell-like mechanisms as

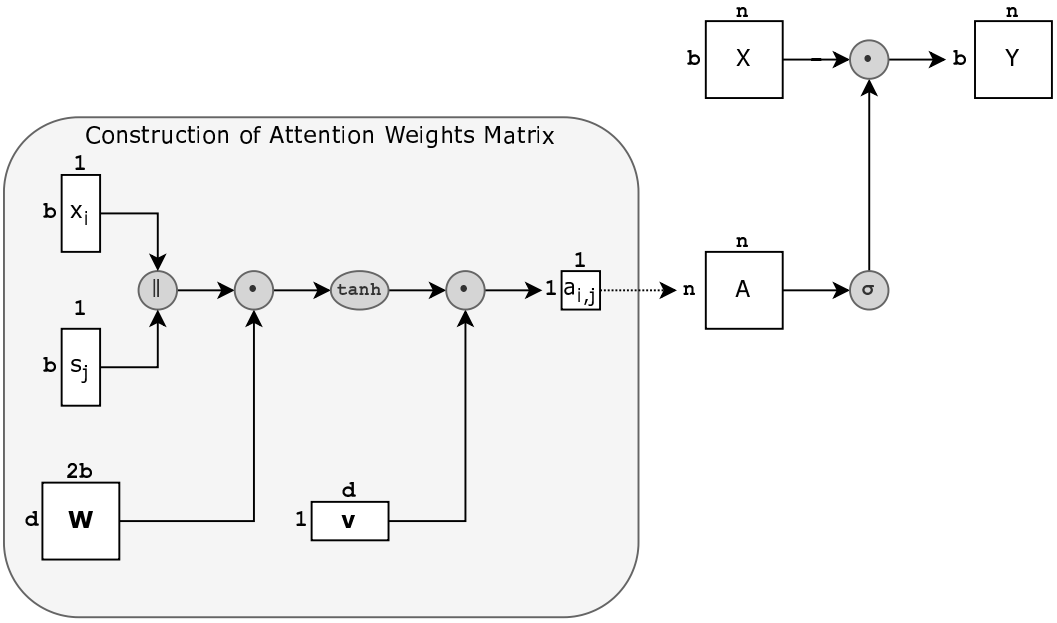


Figure 4.5: A schematic of the matrix operations involved in additive attention. Each element of the attention weights matrix A is derived by passing two elements from the input sequences x_i and s_j through a feed-forward layer. The $\|$ symbol denotes vector concatenation.

exist in LSTMs. However, it requires that a network implementing attention contain some other form of mechanism which can explicitly use information about relative ordering of sequence elements, or the network will treat its input data as an unordered set instead of a sequence.

4.3.1 Query-Key-Value Attention

The dominant variant of attention used in recent years is *Query-Key-Value* (QKV) attention (Vaswani et al. 2017). QKV attention performs separate linear transformations on each input sequence before incorporating them into the model. With trainable parameter matrices $\mathbf{Q} \in \mathbb{R}^{n\times d}, \mathbf{K} \in \mathbb{R}^{m\times d}, \mathbf{V} \in \mathbb{R}^{n\times b}$, the resulting algorithm is:

$$Y = \sigma\left(\frac{\mathbf{Q}X(\mathbf{K}S)^\top}{\sqrt{d}}\right)\mathbf{V}X. \tag{4.23}$$

In Equation 4.23, the intermediate matrices $\mathbf{Q}X$, $\mathbf{K}S$, and $\mathbf{V}X$ are respectively known as the *queries*, *keys*, and *values*, while the trainable parameters $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ represent the *query weights*, *key weights*, and *value weights*. The division by $\sqrt{d}$ inside the softmax function is a heuristic adjustment that improves the training regularity of the model, and it has become a standard part of QKV attention. Within a neural network, Equation 4.23 is referred to as a single *attention layer*. Figure 4.6 shows a schematic of a single QKV attention layer.

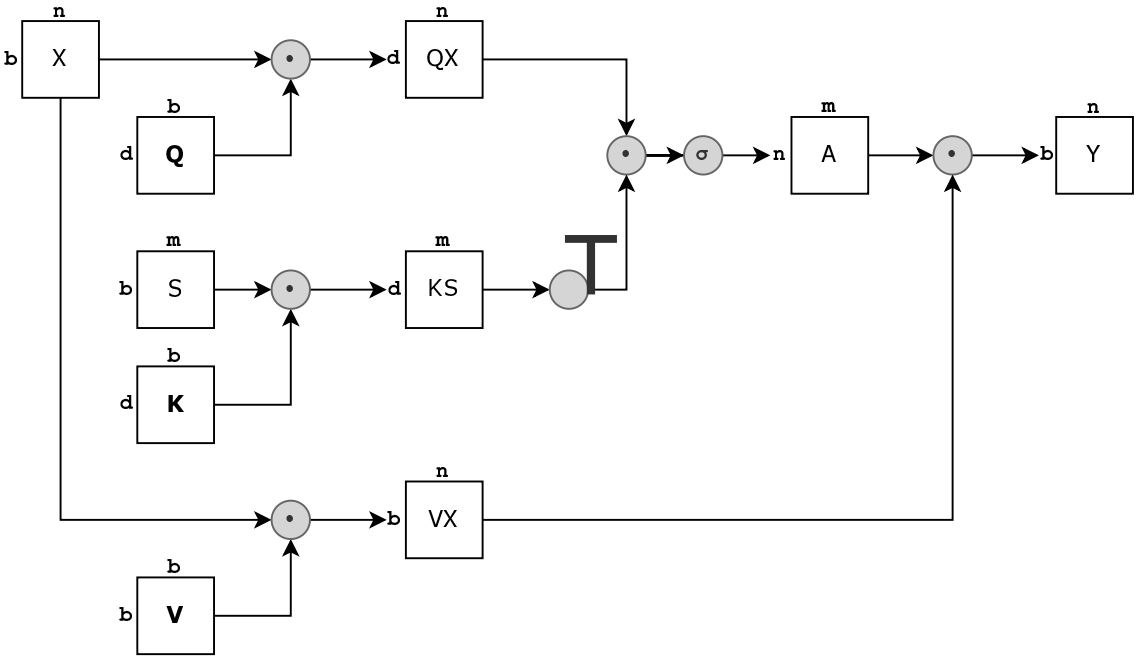


Figure 4.6: A schematic of the matrix operations involved in Query-Key-Value (QKV) attention.

Typically, the value weights $\mathbf{V}$ are chosen to have a shape of $m \times b$, resulting in an output Y with a dimensionality of $n \times b$, matching the input sequence X (recall that the attention weights have a shape of $n \times m$). However, the query and key weights $\mathbf{Q}$ and $\mathbf{K}$ may have shapes of $n \times d$ and $m \times d$, respectively, where d can take any value. Regardless of the chosen d, the product $\mathbf{Q}X(\mathbf{K}S)^\top$ always yields a shape of $n \times m$. As a result, d becomes a hyperparameter of the network known as the *hidden dimension* of the attention mechanism.

4.3.2 Multi-Head Attention

Another widely adopted technique introduced by Vaswani et al. (2017) is the utilization of *multi-head attention*, which involves running multiple attention layers in parallel. The motivation behind this approach is to allow each head to focus on different important aspects of the network input. Empirical analyses of trained weights in multi-head attention-based models for NLP tasks reveal that each head attends to distinct high-level aspects of the input, keeping track of "keywords, locations, organizations, people, or days of the week" (Baan et al. 2019).

The implementation of multi-head attention is straightforward. For h heads, h different weight matrices are employed for each of the queries, keys, and values. Standard attention computations are performed on each set of weights, resulting in h outputs $Y_1 \cdots Y_h$. To combine these outputs into a single sequence of vectors matching the size of the input X, the $Y_1 \cdots Y_h$

outputs are concatenated, and the resulting vector is passed through a feed-forward neural network to reduce its dimensionality back to the original size. I refer to this entire structure, consisting of the attention heads and the final fully connected layer, as a *multi-head attention layer*.

4.3.3 Transformers and Positional Encoding

The *Transformer* network, introduced by Vaswani et al. (2017), is not a mechanism for use in other machine-learning architectures but a stand-alone sequence model that incorporates no recurrent structures whatsoever. I will refer to the original formulation of the Transformer by Vaswani et al., as the *vanilla Transformer*, as is the convention in the literature.

Transformers operate using *self-attention*, which replaces the state sequence S with a duplicate of the input sequence X. A full attention layer for Transformers is:

$$Y = \text{softmax}\left(\frac{\mathbf{Q}X(\mathbf{K}X)^\top}{\sqrt{d}}\right)\mathbf{V}X \tag{4.24}$$

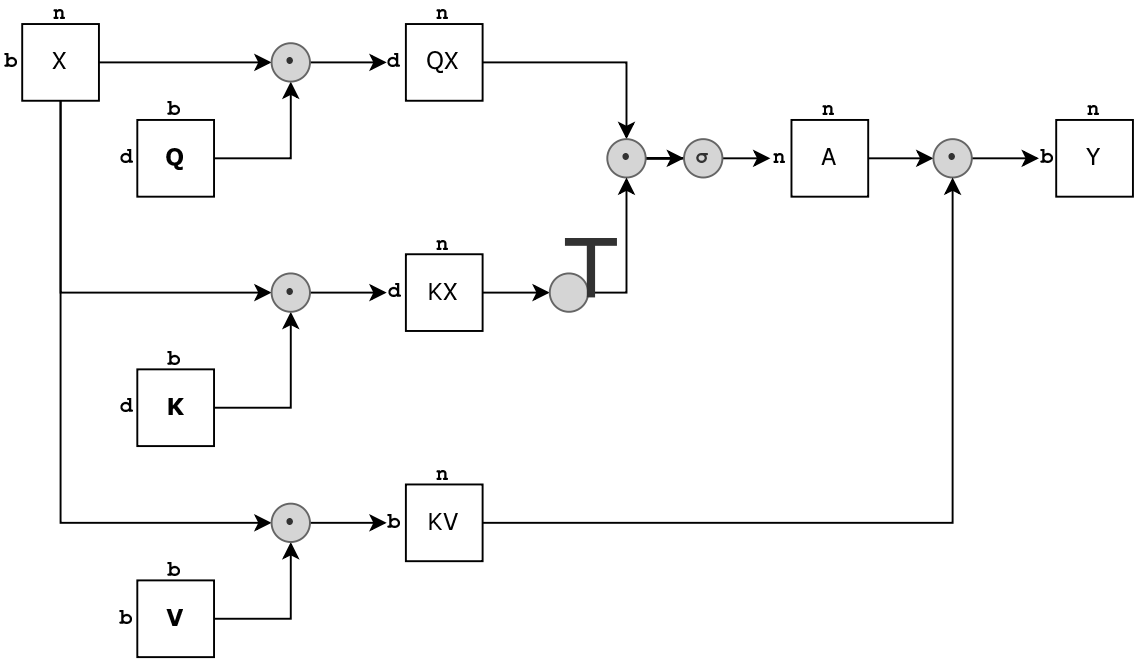


Figure 4.7: A schematic of the matrix operations involved in QKV self-attention, as used in transformers.

Even though the query and key matrices are derived from the same input sequence, it is still beneficial to treat them separately. This distinction allows for non-symmetric attention weights, enabling position i to attend to position j without position j attending to position i.

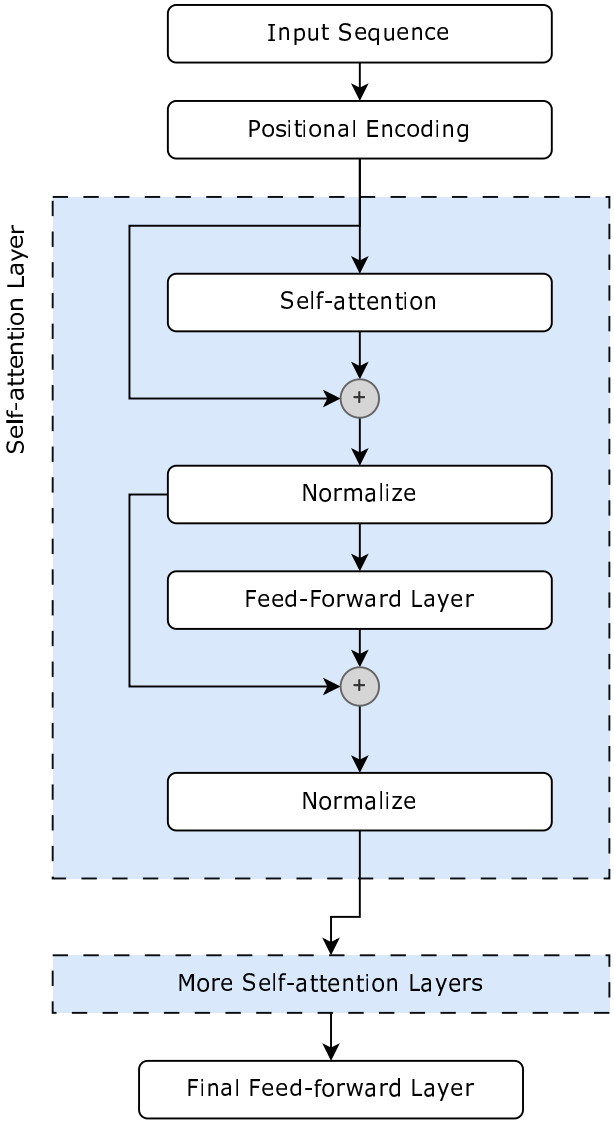


Figure 4.8: A schematic showing the components of the vanilla Transformer.

The vanilla Transformer is composed of multiple multi-head self-attention layers arranged sequentially. Each layer performs multi-head self-attention on an input sequence and passes the processed sequence to the next layer. In the context of a sequence-to-sequence architecture, such as machine translation tasks, two separate Transformer networks are employed. The *encoder* consists of a stack of multi-head self-attention layers that encode the input sentence into a contextualized vector. The *decoder* alternates self-attention layers with traditional two-input attention, utilizing the encoded context vector as the second input. Due to the widespread use of this configuration, a stack of multi-head self-attention layers is commonly referred to as a *Transformer encoder*, even outside of the context of sequence-to-sequence tasks.

Without any kind of recurrent component, the Transformer network has no way of incorporating information on the absolute or relative positions of sequence elements using self-attention

alone. To address this, the network must employ some *positional encoding* method to indicate the position of each element in the sequence. One simple method is to concatenate or sum the index of each sequence element with the element itself (Gehring et al. 2017). However, the designers of the vanilla Transformer proposed a more complex approach that enables the model to learn from relative positional differences between elements. In the vanilla Transformer, the following scheme is used to add positional encodings to an $n \times b$-sized input X in a model with a hidden dimension of d:

$$PE(X_{i,p}) = \begin{cases} \sin\left(\frac{p}{10000^{i/d}}\right), & \text{if } i \text{ is even} \\ \cos\left(\frac{p}{10000^{i/d}}\right), & \text{if } i \text{ is odd} \end{cases} \tag{4.25}$$

The variable i represents the index of each sequence element, while the variable p indexes the d different embedding dimensions. This positional encoding operates by adding sinusoidal functions to each sequence element. Each element in the sequence is combined with a sinusoid that oscillates rapidly for lower values of p and slows down for higher values of p. The starting frequency of the sinusoid is determined by the element's position in the sequence. While there are alternative methods of positional encoding, such as *relative positional encoding* (Pham et al. 2020; Shaw et al. 2018), the sinusoidal-based positional encoding remains the most widely adopted approach in practice. Figure 4.7 illustrates QKV self-attention, and Figure 4.8 shows how these attention blocks are assembled into the Transformer architecture along with a positional encoding method.

4.3.4 Efficient Transformers

Attention mechanisms, including those used in Transformer networks, offer distinct advantages over recurrent networks in their ability to combine information from elements of a sequence that are distant from each other. This advantage arises from the attention weights that require the computation of a similarity score between every element in the input sequence and every element in the state sequence (or every pair of elements in a single sequence, in the case of self-attention). This full pairwise comparison is computationally expensive both to perform and to store in memory, impacting the memory usage during both training and inference stages.

During backpropagation, the backward pass in recurrent neural networks has a time and space complexity of $\mathcal{O}(n)$, where n represents the length of the input sequence. This efficiency is achieved because the hidden states have a fixed size, and the backpropagation-through-time

algorithm only needs to store a fixed number of partial derivatives for each timestep. On the other hand, attention mechanisms require the creation of an $m \times n$ attention weight matrix, where m and n represent the lengths of the input and state sequences, respectively. Consequently, the computational complexity of backpropagation in both space and time for Transformers is $\mathcal{O}(n^2)$ with respect to the input length, making it infeasible to apply the vanilla Transformer to very long sequences. As a result, there has been extensive research aimed at developing memory-efficient variants of the Transformer architecture.

The field of efficient Transformers is developing rapidly. A review by Tay et al. (2023) found thirty-two novel methods for improving the computational and memory complexity of the vanilla Transformer published in the years between 2018 and 2022. For the remainder of this section, I review the broad categories of efficient Transformer, along with notable examples of each kind.

4.3.4.1 Fixed Pattern Attention

The earliest efforts to make Transformers more efficient approximated the full attention weight matrix by simply limiting the parts of the matrix that were calculated, a method called *fixed pattern attention*. This was first implemented by Parmar et al. (2018) in their *Image Transformer* which partitions the keys and queries into smaller blocks that are less computationally intensive to multiply together, assembling a set of attention weights in a piecewise manner; the resulting complexity is $\mathcal{O}(nm)$, where m is the size of the smaller blocks. Another implementation of this idea is *sparse attention*, introduced by Child et al. (2019), where attention weights are calculated for a different, disjoint subset sequence positions for each self-attention head. This reduces the complexity of the operation $\mathcal{O}(n\sqrt[d]{n})$, where d is the number of self-attention heads.

Another notable variant in this category is *FNet*, proposed by Lee-Thorp et al. (2022), which utilizes the Fast Fourier Transform (FFT) in place of self-attention. The part of the Fnet corresponding to the standard multi-head attention layer contains no trainable parameters; the network consists of standard feed-forward fully connected layers interleaved with FFT "mixing" layers. Despite the absence of trainable parameters in the FFT, the FNet exhibits only a marginal decrease in performance compared to vanilla Transformer models, typically within a few percentage points. Additionally, the FNet demonstrates efficient scalability to long sequences, as the time and space complexities of FFTs are only $\mathcal{O}(n \log n)$ with respect to the sequence length.

4.3.4.2 Learned and Random Pattern Attention

The related technique of *learnable pattern attention* calculates only a limited number of entries in the attention weight matrix, but dynamically learns which ones to calculate during training. A Transformer variant implementing this strategy called the *Reformer*, proposed by Kitaev et al. (2020), takes advantage of the importance of the largest entries in the self-attention matrix. These entries indicate highly similar elements between queries and keys, playing a crucial role in capturing meaningful relationships. By reformulating the attention mechanism as a nearest-neighbors task, solvable using locality-sensitive hashing, they are able to find these largest entries without performing a full matrix multiplication. The Reformer achieves a time and space complexity of $\mathcal{O}(n \log n)$ in the input length, though it requires query weights and key weights to be identical for the nearest-neighbor search to be possible.

Tay et al. (2020) proposed an approach called the *Synthesizer* as an alternative to dot-product attention. Rather than relying on queries and keys to compute attention weights, the Synthesizer directly constructs the attention weight matrix from the input sequence by passing it through a series of feed-forward layers. The values are then calculated using a trainable matrix of value weights and multiplied with the attention weights as with normal QKV attention. The authors demonstrate that Synthesizers outperform vanilla Transformers in terms of performance on common benchmarks and training time. Additionally, they introduce the *Random Synthesizer*, which uses a fixed, randomized attention weight matrix that is independent of the input. Remarkably, the Random Synthesizer achieves comparable or better performance than the vanilla Transformer across a range of tasks.

4.3.4.3 Low-Rank and Kernel Attention

Instead of approximating the self-attention matrix, some methods seek to algebraically rearrange the self-attention mechanism to avoid directly computing it at all. Consider the QKV formulation of attention stated in Equation 4.23, here reproduced with annotations denoting the size of each vector:

$$\mathbf{Y} = \sigma\left(\frac{1}{\sqrt{d}}\underset{n\times d}{\mathbf{Q}X}\ \underset{d\times n}{(\mathbf{K}S)^\top}\right)\underset{n\times b}{\mathbf{V}X} \tag{4.26}$$

The matrix multiplication between $\mathbf{Q}X$ and $(\mathbf{K}S)^\top$ is the step that causes the quadratic complexity. If the softmax function were removed, the equation could be rewritten as shown in

Equation 4.27, where the intermediate matrix is of shape $d \times b$:

$$\begin{aligned} Y &= \frac{1}{\sqrt{d}} \underset{n\times d}{\mathbf{Q}X} \; \underset{d\times n}{(\mathbf{K}S)^\top} \; \underset{n\times b}{\mathbf{V}X} \\ &= \frac{1}{\sqrt{d}} \underset{n\times d}{\mathbf{Q}X} \; \underset{d\times b}{\left[(\mathbf{K}S)^\top \mathbf{V}X\right]} \end{aligned} \tag{4.27}$$

By first multiplying the keys and values, the intermediate matrix is no longer of shape $m \times n$ but of shape $d \times b$, resulting in memory requirements of $\mathcal{O}(n)$ in the sequence length. In this case, the memory usage is primarily determined by the dimensions of the token embedding b and the internal hidden dimension d. Achieving this in a vanilla Transformer is not possible due to the nonlinearity of the softmax function, preventing the rearrangement of terms. A primary focus of research into increasing the efficiency of Transformer networks is to find methods of approximating, replacing, or otherwise bypassing this softmax nonlinearity so that the query-key multiplication can be avoided altogether. Approaches that take this strategy are said to use some form of *linear attention* or *low-rank attention*.

Shen et al. (2021) proposed applying the softmax function in two separate places:

$$\mathbf{Y} = \frac{1}{\sqrt{d}} \, \sigma_{\text{row}}\left(\mathbf{Q}X\right) \sigma_{\text{column}}\left(\left[\mathbf{K}S\right]^\top\right) \mathbf{V}X. \tag{4.28}$$

Here, σ_{column} denotes a column-wise softmax and σ_{row} a row-wise softmax. While this is not equivalent to the softmax in Equation 4.23, the authors proved that it is similar in qualitatively important ways. In particular, the intermediate matrix $\sigma_{\text{row}}\left(\mathbf{Q}X\right) \sigma_{\text{column}}\left(\left[\mathbf{K}S\right]^\top\right)$ has the property that each row sums up to one, and therefore it represents a valid set of probability distributions, but it does not need to be explicitly computed.

Katharopoulos et al. (2020) proposed a method to bypass the softmax nonlinearity by approximating it using the *kernel trick*. This technique, commonly employed in machine learning, approximates the nonlinear similarity function $s(x, x')$ between two elements in a feature space by the dot product in another space:

$$s(x, x') \approx \langle \phi(x), \phi(x') \rangle. \tag{4.29}$$

Here, ϕ represents a *kernel* function, which approximates the softmax result after the query-key multiplication. By replacing the softmax function with standard matrix multiplication in a

different space, they eliminate the need for the softmax operation on the keys and values. Since the transformation is linear, the keys and values can be multiplied together before applying the kernel, as shown in Equation 4.26. The implementation of their linear attention can be expressed as:

$$Y = \frac{1}{\sqrt{d}} \underset{n \times c}{\phi(\mathbf{Q}X)} \left[\underset{c \times n}{\phi\left(\mathbf{K}S\right)^{\top}} \underset{n \times b}{\mathbf{V}X} \right]. \tag{4.30}$$

Here, c represents the dimensionality of the output space of the kernel function. For their kernel, the authors utilize the exponential linear unit (Clevert et al. 2016), a simple nonlinearity. Although the induced similarity function differs qualitatively from the softmax function, their method achieves comparable accuracy to the vanilla Transformer during training while significantly reducing memory requirements.

4.3.4.4 Attention with Memory or Recurrence

Rather than improve the efficiency of the self-attention mechanism itself, alternative approaches have emerged that incorporate recurrent mechanisms. These methods propose performing full dot-product self-attention within smaller regions of the input sequence and utilizing recurrent connections to enable information exchange between adjacent regions. This idea forms the foundation of *Transformer-XL,* introduced by Dai et al. (2019). Transformer-XL computes multi-head self-attention on fixed-length segments of the input and generates both a hidden state vector and an output. It employs a modified version of self-attention that attends to the hidden state in addition to the input, enabling it to retrieve information from previous segments of the sequence, similar to how LSTM operates. This modification significantly expands the learning capacity of the Transformer, allowing it to capture longer-range dependencies than the vanilla Transformer, as well as being orders of magnitude faster at evaluation time. Another technique that incorporates memory into the Transformer framework involves calculating a context vector over the entire input, which can be attended to by other attention operations within the sequence. This approach was initially employed by Lee et al. (2019) in their *Set Transformer.*

In their comprehensive review, Tay et al. (2023) note that these strategies for efficient attention are orthogonal to those focusing on modifying the construction of the attention weight matrix, and observe a growing trend in the development of efficient Transformer models that combine multiple techniques discussed in this section. These models integrate both memory-

based approaches and low-rank or learned-pattern methods within the same Transformer variant, resulting in further improvements in efficiency and performance.

4.3.5 The Long Short-Term Universal Transformer

While the vanilla Transformer is capable of learning long-range dependencies with no restriction on the temporal gap between related tokens, the generality of QKV attention also gives it some weaknesses. RNNs are, strictly speaking, not able to model as many possible phenomena, but the process of continually modifying a hidden state lends itself well to modelling processes where individual sequence elements are closely related to those close to each other in the input sequence. A more formal term for this is that RNNs have an *inductive bias* towards learning certain types of transformations in their training data due to the limited nature of their architecture.

Citing these limitations and several failures of the vanilla Transformer on tasks that LSTMs excel at, Dehghani et al. (2019) propose a modification to the Transformer architecture called the Universal Transformer (UT). This architecture makes a single but important change to the Transformer architecture; instead of passing through several distinct multi-head attention layers, the UT employs a single layer, and uses it to process the same input multiple times, iteratively refining a single input into a target representation. Alternatively, this can be interpreted as a multi-layer transformer where all the trainable parameters are shared between each layer, and so the layers remain identical during backpropagation and SGD. This limitation introduces a new inductive bias into the architecture that the authors demonstrate leads to significantly improved performance on several sequence modelling benchmark tasks. The number of times that input is re-input into the model (or, the number of coupled multi-head attention layers, in the alternate interpretation) is referred to as the *recurrence depth* of the UT. This architectural modification is orthogonal to any of the efficient attention variants discussed in Section 4.3.4, so it can be used in combination with techniques to reduce the computational complexity of QKV attention.

The final Transformer variant that I discuss here is the LSTUT, a variant introduced by Berardinis et al. (2020) for the purpose of modelling long-term dependencies in symbolic music in particular. Recognizing the relatives strengths of LSTMs and the UT, the authors propose an architecture that consists of a single UT bookended by bidirectional LSTMs. The positional encoding mechanism used in all other forms of Transformer network is discarded, as the LSTM networks do have an inherent sense of the ordering of the inputs, and they should be able to convey information about order to the Transformer component on their own. The authors

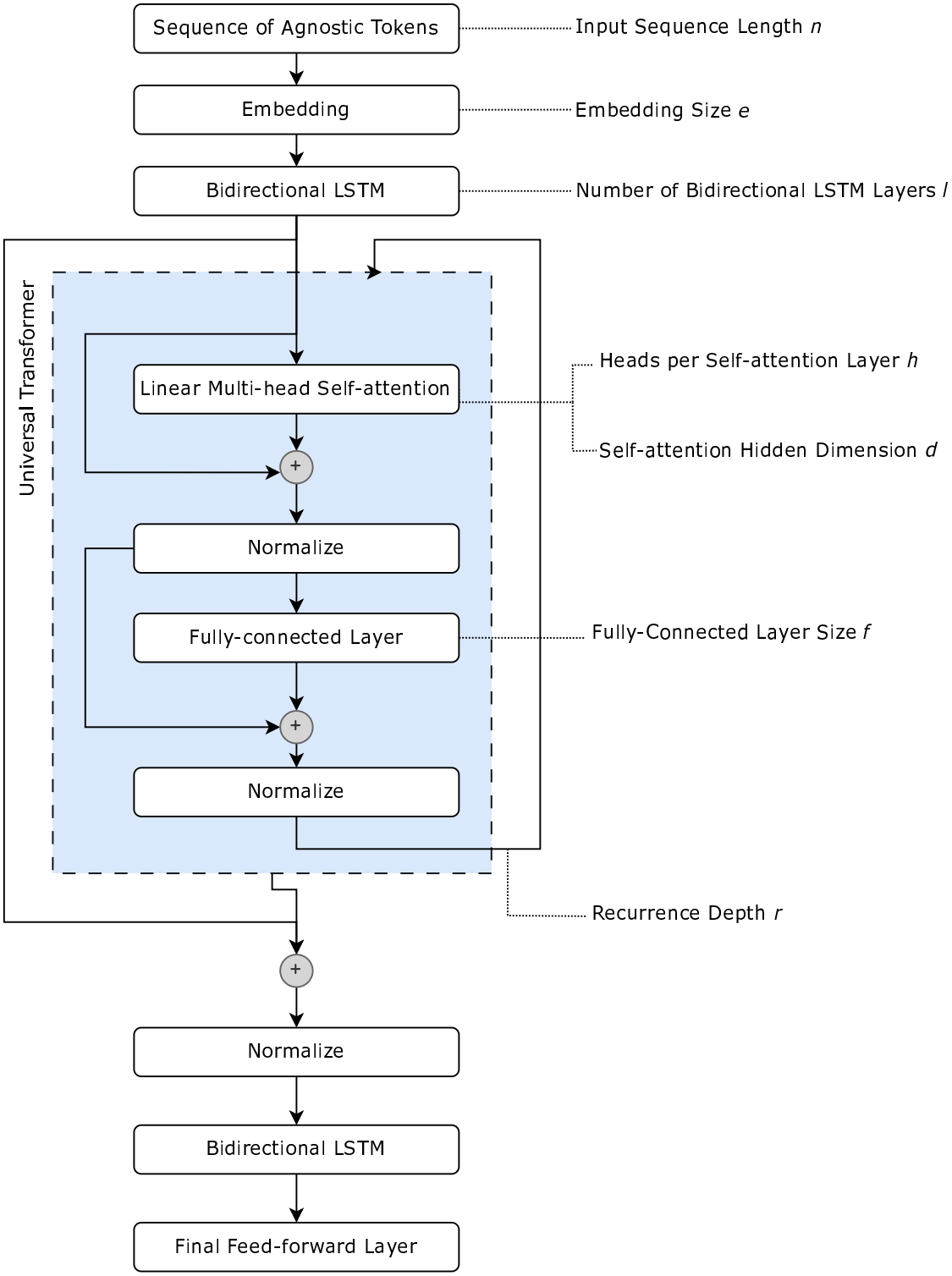


Figure 4.9: A schematic of the Long Short-Term Universal Transformer (LSTUT) network that I use for the error detection task. Modifiable parameters of the network architecture are listed on the right, connected to the parts of the network that they modify. Activation functions (present after every fully connected layer, LSTM layer, and self-attention layer) are omitted from this diagram for the sake of concision.

motivate the design of this architecture by hypothesizing that the LSTMs improve the ability of the architecture to learn local structures of the music, including phrase-level repetition and

texture. This allows the Transformer component to focus on longer-range repetition, which is the aspect of music modeling that LSTMs traditionally perform poorly at. Figure 4.9 shows a schematic diagram of the LSTUT architecture.

I use a version of the LSTUT as the main architecture in the experiments of Chapter 6. I choose it over other models simply because it is the only variant of the Transformer architectures that I am aware of that was designed specifically for use in long-term dependencies in polyphonic symbolic music, and has shown promise in that regard in experiments and in qualitative evaluations. Other methods of adding inductive biases to QKV attention-based architectures typically base their evaluations on benchmarks involving processing of real-valued signals or NLP.

4.4 Machine Learning and Long-Term Dependencies in Music

Learning and reproducing long-term dependencies in sequence data has long been recognized as a challenging task in machine learning, even when employing architectures explicitly designed to handle such dependencies. The issue of long-term dependencies is frequently cited as an explanation for the qualitatively unsatisfying outputs of generative algorithms across various fields. To close this chapter, I discuss the problem of long-term dependencies in machine learning, with a specific emphasis on its significance in the context of modelling symbolic music. I explore the historical definition of the problem and discuss domain-specific techniques that have been utilized to address it.

4.4.1 Qualitative Descriptions of the Problem of Long-Term Dependencies

Here, I present diverse examples of practical deficiencies in generative algorithms to illustrate the ubiquity of the problem. Although music generation is not the main focus of this thesis, it is a domain where failures to capture long-term structure are well-documented; I hypothesize that such failures are simply more evident in music generation than in other applications. These deficiencies are occasionally noted in other areas as well; for instance, in the field of Automatic Music Transcription (AMT), Boulanger-Lewandowski et al. (2013) found that while modelling temporal dependencies was beneficial, capturing longer-term musical structure remained elusive in their approach based on recurrent neural networks. In addition, my error-checking method employs a Transformer architecture similar to many of the papers discussed above. A fundamental assumption of my approach is that long-term structure in music often exhibits redundancies

in an informational sense, and that these redundancies could aid in identifying musical errors. If Transformer architectures struggle to learn from long-term structure in generative tasks, it is reasonable to expect the same challenges in an error-correction task as well, and I must consider and address these challenges in the design of my method.

In the early stages of using neural networks for music generation, Mozer (1994) employed simple recurrent networks, which yielded results with sensible local contours. However, these outputs were "not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organization." Early successes were achieved by focusing on corpora of music with simple and strictly standardized structures. For instance, Eck and Schmidhuber (2002) trained an LSTM to generate music following the twelve-bar blues structure. While the generated melodies resembled real, if "not very good" jazz bebop improvisations, there was no variation in formal structure or chord progression among the training examples. Similarly, Sturm et al. (2016) trained an LSTM on a large corpus of monophonic Irish folk melodies and successfully recreated the simple AABB structures commonly found in the dataset. The authors attribute this success to the dataset size, the regularity of the learned structures, and the increased size of the LSTM models compared to those used in previous research.

The use of Transformer networks in music generation was pioneered by the Music Transformer (Huang et al. 2019), specifically designed to capture large-scale musical structure. However, the quadratic memory requirement posed a challenge when dealing with long musical sequences exhibiting structural complexities. To address this, Huang et al. developed a memory-efficient variant of attention tailored for music generation. They reported the ability to generate repeated patterns on a scale of minutes, surpassing the time span achievable by LSTM-based methods that typically reached tens of seconds. Since then, there have been numerous studies exploring music generation using Transformers. In a comprehensive review conducted by Civit et al. (2022), the authors observed that Transformer-based approaches focused on musical structure successfully generate coherent music in the short- or medium-term, and could serve as potential foundations for new compositions. However, they emphasize that these methods still lack consistency in the long-term, requiring some degree of human editing to produce songs of human-composed quality.

To generate music with structure, a common approach is to incorporate domain-specific adaptations that explicitly encode structural rules into the generation process. One such approach is StructureNet, introduced by Medeot et al. (2018), which generates melodies hierarchically using an LSTM. It first generates an overall repetitive structure for a piece and then fills it in

with musical material. Another strategy, presented by Jhamtani and Berg-Kirkpatrick (2019), involves creating a differentiable measure of self-similarity. They train a generative model to generate music that matches both a given corpus and the corpus's self-similarity characteristics. Other related strategies, outlined in a review by Ji et al. (2023), include representing music as hierarchical tree structures and employing neural networks that operate on graphs. Conditional generation is also commonly used, where music is generated to adhere to a chord progression or bassline provided by the user or generated by a separate network. These methods often operate on transformed or simplified versions of music, making the long-term structure more explicit and side-stepping the need to learn complex long-term dependencies. However, they rely on prior knowledge of the expected structure and require reliable algorithms to automatically identify it. As a result, these approaches are not general-purpose and are typically fine-tuned to operate on specific genres and instrumentations.

Despite the wealth of research on music generation, there is a lack of quantitative assessment regarding the extent to which the structures of generated music deviate from those found in real music. To address this gap, Dai et al. (2022) conducted an analysis to quantify how the outputs of the Music Transformer and other machine-learning music generation techniques differ from the usual organizing principles of human-composed music. They identified four distinct tendencies related to repetitive structure in music that could be measured statistically: multiple hierarchical levels of repetition, a limited musical vocabulary within individual songs, the interaction of song structure with melody and rhythm, and the amount of repeated content over the course of a song measured using information-theoretic metrics. In human-composed popular music, these statistical measures exhibit clear trends and are confined to specific ranges. However, in music generated by Transformers, these measures show less constraint and deviate from predictable trends. For instance, generated music tends to utilize a more varied and less predictable melodic vocabulary within individual pieces compared to human-composed music.

4.4.2 Formal Treatments of Long-term Dependencies

Despite the consensus on the difficulty of handling long-term dependencies in machine learning, there has been limited research exploring the formal aspects of "the ability to learn from long-term dependencies" on an algorithmic level. How can one formally assert that one architecture is inherently better at long-term tasks than another? Zhao et al. (2020) observed the challenge of making formal claims about the behavior of a network architecture based solely on heuristic

assessments of its performance on tasks involving long-term reasoning. In this final section, I review the formal analyses conducted in this area.

To investigate the properties required for a recurrent sequence model to effectively learn from data with long-term dependencies, Bengio et al. (1994) proposed a minimal set of criteria:

- The system must exhibit *information latching*, enabling the storage of information for an arbitrary length of time.
- The system must be resilient to noise, capable of distinguishing between useful and irrelevant information for making predictions.
- The parameters of the system must be trainable via gradient descent and there must be reasonable guarantees of convergence in a finite number of timesteps.

However, the authors discovered that, for recurrent networks, the use of gradient descent itself impedes the system's ability to latch information and be robust to noise. Specifically, the conditions necessary for stable and convergent gradient descent optimization cause the gradients to exponentially approach zero during training if information latching is required. LSTMs, which outperform traditional RNNs in practice, show promise in synthetic benchmarks designed to necessitate long-distance information latching (Hochreiter and Schmidhuber 1997b) and mitigate the problem of vanishing gradients during training (Hochreiter 1998). Nevertheless, Cheng et al. (2016) demonstrated that LSTMs still fall within the class of gradient descent-trained recurrent methods, implying fundamental limitations on their capacity to latch information over long sequences.

In another approach to formalizing the problem of long-term dependencies, Greaves-Tunnell and Harchaoui (2019) incorporated the concept of *long memory*, a statistical property of time-series sequences unrelated to the "long short-term memory" after which the LSTM network is named. Long memory in a sequence is characterized by a slow decay of *autocovariance*, which measures the covariance between the sequence and its time-shifted copies as the time-shift increases. The authors rigorously define this slow decay and use it to classify different models based on their ability to represent sequences with long memory. Their analysis revealed that common models, including Markov chains, moving average models, and RNNs, were unable to effectively represent sequences with long memory. In empirical tests on both English text and musical datasets, they found that a wide variety of text datasets all exhibited long memory, with music datasets showing an even stronger presence of long memory than text. Building upon

these findings, Zhao et al. (2020) established that LSTMs also lacked the capability to model sequences with long memory. They proposed architectural modifications, similar in design to standard attention mechanisms, that could enable LSTMs to capture long memory.

To my knowledge, there is no research specifically analyzing whether vanilla Transformers or their variants possess the ability to model sequences with long memory, as has been formally established for RNNs and LSTMs. Although Transformer-based architectures have been extensively evaluated on standardized or synthetic tasks involving long-term dependencies, such as the Long Range Arena (Tay et al. 2021), their impressive performance on these tasks does not necessarilly translate to the capability of modeling long memory. Notably, despite their capabilities, vanilla Transformers are not Turing-complete like RNNs are, and struggle with simple tasks such as text copying that LSTMs handle with ease (Dehghani et al. 2019).

The question of whether introducing architectural changes to Transformers could equip them with the ability to learn long memory in a manner beneficial for modeling symbolic music falls beyond the scope of this dissertation. However, based on the existing research, it can be concluded that there is no known "silver bullet" solution that enables Transformer networks to learn coherent rules of musical structure. The machine-learning approaches employed to capture musical structure in symbolic music still rely on domain knowledge, data curation, and data preprocessing, often at the expense of generalizing across different types of musical structures.

Chapter 5

Methodology

In this chapter, I outline the algorithmic framework for the musical error detector, a system designed to speed up the process of correcting errors in Optical Music Recognition (OMR) output. This system consists of a machine learning-driven application that processes hierarchically structured, symbolic music files and generates versions annotated with predicted error locations. To train this model, I assemble a dataset of string quartets, alongside a data augmentation strategy to effectively enlarge the training dataset. Additionally, I create a method for transforming polyphonic symbolic music into one-dimensional sequences of agnostic tokens suitable for input into a machine-learning sequence model. The evaluation of this system is detailed in Chapter 6.

Section 5.1 describes the intended capabilities of the error detector. I discuss the underlying motivations for its design choices and how these aims shape its overall architecture. Section 5.2 examines how various representations of music influence the definition and identification of errors. This analysis has implications for the error detector's architecture, emphasizing the need for a tailored approach to error detection in music. I introduce a method for converting polyphonic, multi-staff scores into one-dimensional sequences of agnostic tokens. A procedure for identifying errors within these sequences is established using the Affine Needleman-Wunsch (ANW) algorithm, setting the groundwork for the error detector's operational logic. Section 5.3 describes the composition and characteristics of the dataset of string quartets compiled for training the model. Given the scarcity of data containing OMR errors with corresponding corrected versions, I define various strategies for creating synthetic OMR errors that facilitate training the error detector under simulated conditions. Section 5.4 presents the Transformer-based neural network model selected for the error detector and also details the hyperparameter optimization process undertaken to fine-tune the model's architecture. Finally, in Section 5.5 I

summarize the chapter and detail how the described components are assembled into training, testing, and inference procedures.

5.1 Motivation

Upon completion, the error detection system presented in this chapter aims to identify incorrectly recognized symbols resulting from the OMR process. A detailed explanation of the term "musical symbol" is provided in Section 5.2.3, where I describe the method I use to break down a semantically encoded polyphonic score into a one-dimensional sequence of tokens.

The system, from an end user's perspective, should function as follows: the user provides a file containing symbolic music in MusicXML format, and the system outputs an altered version of this file. The program outputs a version of that file which is identical except that individual tags of the file (representing notes, symbols, markings, etc.) have been marked with a color where the method predicts that the tag represents an erroneously predicted glyph on the page. The system would feature a singular user-adjustable parameter to modulate its sensitivity, allowing the user to influence the number of glyphs that are marked as errors. This aligns with the concept of threshold setting in binary classification, detailed in Section 4.1.2. This tool's primary purpose is to support human review and subsequent manual corrections of scores processed through OMR. The design will exclusively focus on this use-case, without considering potential alternative applications such as assisting in audio-to-score transcription corrections or any theoretical educational or creative uses.

I here present a list of design goals for the error detector:

- **Genre and instrumentation agnosticism**: While the model will initially be trained on a specific musical genre for evaluative purposes, it is imperative that the method itself is not intrinsically bound to the characteristics, tendencies, or instrumentation of any single musical genre. The model should be adaptable to any genre that utilizes Western music notation, regardless of the instrumentation, musical texture (homophonic, heterophonic, polyphonic), or the number of distinct voices on a single staff.

- **Prioritizing high recall rate**: The principal aim of the error-detection approach is to minimize false negatives. The model should identify every error present in the input. A higher rate of false positives is deemed acceptable if it ensures that no errors are overlooked, thus providing users with confidence that all potential errors have been flagged.

- **Granular error localization**: The detector should function with precision at a granular level, pinpointing individual glyphs as erroneous rather than flagging entire measures or systems. This granularity is crucial for detailed and accurate error identification.

- **Feasibility with limited OMR error data**: In an ideal scenario, the error correction system would be trained on a large corpus of music processed through OMR, each accompanied by a corrected, ground-truth version. However, such extensive datasets do not exist in publicly available forms, and generating them at scale is not feasible with commercial OMR systems. Therefore, the method must be effective even with a smaller dataset of OMR errors, supplemented by data augmentation techniques similar to those used in natural language error correction and detection tasks (see Section 2.2.1).

- **Adaptability to small music corpora**: Given that many genres commonly processed by OMR lack extensive, high-quality digital datasets of corrected music, the error detection system must be effectively trainable on smaller music corpora. The capability to enhance limited datasets through data augmentation is essential. The system should maintain reasonable performance levels, even in scenarios where extensive digital datasets in the specific genre are unavailable to use for data augmentation.

- **Ability to process long inputs**: As highlighted in Section 1.2, music inherently exhibits a higher degree of repetitiveness compared to natural language, often featuring long-term redundancies. Consequently, OMR-induced errors might not be discernible in isolation but only apparent when viewed in the context of larger music sections or the entire piece. The error detection method should, therefore, be equipped to handle long inputs, enabling it to analyze significant portions or even complete musical pieces in one go. This ability is crucial for accurately identifying errors that depend on broader musical context. For example, a stylistically uncommon vertical harmony may or may not be considered erroneous depending on whether it recurs in other parts of the piece.

In the current error detection task, I refrain from classifying distinct musical error types, a practice occasionally seen in Grammar Error Detection (GED) (see Section 2.2). A primary reason for this is the challenge of formalizing high-level musical error descriptors, though they can be inferred from music theory literature. The intricacies of classifying errors within the OMR evaluation scope are discussed in Section 3.2.2. This is distinct from natural languages

where errors, such as prepositional and subject-verb agreement, are well-documented and collectively recognized by the field of Natural Language Processing (NLP). While it might be possible to categorize errors into a limited set of low-level classes, indicating deletion, insertion, or replacement of specific glyphs relative to the ground truth, preliminary experiments found this formulation of the problem to consistently degrade the overall performance of the method.

Separating errors into distinct categories might still assist human correctors in reviewing OMR outputs, but whether this advantage outweighs the cost of decreased performance is uncertain without empirical evidence from a user-focused study. Drawing from my own encounters with OMR output, I hypothesize that this would not be the case. When an error is spotted in a score, either via the error detector or through the corrector's own musical intuition, one typically cross-references against the original scanned score to check the correct rendition. This will always reveal the nature of the error and the corrective measures required. Thus, explicitly providing this information in the detector's results could be superfluous, benefiting only in specific instances where the resolution of the error can be deduced solely from the error detector's output. In contrast, in GED contexts, error detectors usually serve pedagogical applications. Clearly indicating the rationale behind each error classification in these cases can be highly instructive. Consequently, for this project, I limit the scope to binary classification, categorizing each symbol as either erroneous or correct. Potential future avenues for implementing a more detailed error detector are discussed in Section 7.3.2.

5.2 Representing Errors in Polyphonic Scores

How does one define an "error" in a musical score? In this research, I take the view that a human-reviewed and fully corrected transcript of a printed score can be taken as ground truth as a faultlessly accurate representation of a musical piece, and that any deviation from that human-reviewed score must constitute an error. Defining the notion of one score deviating from another requires some procedure to take a difference between two scores (in this case, between an OMR output and a corrected score). The characteristics of the errors displayed to the user of the musical error detector will depend on what kind of difference procedure is used and what underlying musical representation it employs.

5.2.1 Considerations in Choosing a Representation and Difference Operation

This subsection deals with the question of how to translate a polyphonic, hierarchically organized file of symbolic music into a form appropriate for both input into a machine-learning model and conducive to a workable difference operation.

I consider the following criteria in choosing a representation:

- **Granular.** Given the aim of pinpointing errors with high precision, it is advantageous for each individual unit of representation to hold minimal information. This ensures that when identifying an element as erroneous, the error is localized to the smallest possible segment of the score.

- **Low Vocabulary Size.** If the representation defines a large number of possible tokens, it increases the odds of encountering out-of-vocabulary data in test sets or during inference. This effect is exacerbated by the relatively small dataset I use; many possible tokens would not be present in the training data for the error detector. The presence of out-of-vocabulary tokens has been found to significantly degrade the performance of related language models on NLP tasks (Moon and Okazaki 2021).

- **Ability to represent OMR errors.** The representation must be capable of representing purely notational errors that do not affect the sound of the music when performed (e.g., errors in beaming, enharmonically equivalent notes, or stem direction).

- **Suitability for concisely representing OMR errors.** Different representations may represent the same difference between scores with different numbers of errors. For example, a deleted accidental may be interpreted as causing many errors, since it affects the pitch of subsequent notes, or it may be interpreted as only being a single error, since only one symbol was changed. Often, OMR techniques are grounded in the misidentification of primitive shapes, and this tendency influences the kinds of inaccuracies induced into musical scores (see Table 3.1 for a list of common OMR errors). The representation chosen should aim to represent the kinds of errors introduced by OMR as concisely as possible.

- **Suitable for use in a sequence alignment algorithm.** In their simplest forms, methods of finding differences between sequences of elements such as the ANW algorithm only address scenarios where sequence elements are either identical or different, with no possibility for partial similarity. Using a representation that assigns multiple values to each

symbol would require producing a custom set of weights for the algorithm to define when partial matches are desirable between two elements. It is not clear how such a weighting system should be designed, or how to judge that one weighting system to be more appropriate for the task than another. Conversely, employing a token-based system, where every element consists only of a single categorical value, establishes a direct link with the extensive existing literature on Grammar Error Correction (GEC) and GED that uses this kind of representation (see Section 2.2).

- **Error shape compatible with machine learning.** The representation must admit an error representation compatible with a machine-learning model's requirements. The "shape" of the error, when represented as data, should remain consistent across all potential errors. For instance, the Feed-forward Neural Network (FNN) (Section 4.2.1) and the Transformer decoder (Section 4.3.3) require as input sequences of a specific length, where every element of the sequence is a real-valued vectors with a specific dimensionality.

5.2.2 Potential Musical Representations

In evaluating common symbolic music representations for training the error detector, several existing formats are not suitable. Musical Instrument Digital Interface (MIDI) files, for instance, lack detailed notational information. Piano roll-like representations, which depict music as a two-dimensional array, are similarly limited. The Humdrum `**kern` format, despite containing more information, fails to uniquely specify certain score elements like beaming configurations and is therefore also not ideal for this purpose.

A potential candidate for musical representation is the *tuple-like* form, such as NoteTuple (Hawthorne et al. 2018). This format represents polyphonic music as an ordered series of multi-valued tuples, with each tuple element representing different musical attributes like duration, pitch, beaming, stem direction, and time to the next event.[1] This approach could adapt to various notational phenomena by varying the number and type of tuple elements. Tuple-like representations are common in music deep learning applications; however, this approach has limitations for our purpose. However, using a tuple-like kind of representation would require defining a notion of difference for notes that vary in certain attributes but not others. Notions of sequence alignment like the ANW alignment (Section 2.4.3) would require special adaptation to

1. Here, the `time to next event` and `duration` entries differ only when encoding polyphonic music, where it can be necessary to encode multiple notes that sound simultaneously but have different start or end times. In monophonic music, these two entries are always the same.

this representation. This would also add an extra layer of complexity for the machine-learning model to learn.

Hierarchically structured musical data formats like MusicXML and Music Encoding Initiative (MEI) are adept at representing the complexities of polyphonic music. They can be directly input into specialized deep learning models, such as graph neural networks, which have been used in some recent studies for symbolic music analysis (Karystinaios and Widmer 2022) and to aid OMR (Baró et al. 2022). These formats naturally suit the structure of musical scores and are thus inherently suitable for representing detailed musical information. However, when I began this research, deep learning architectures designed to capture long-term dependencies in graph-structured data were not extensively researched, and their implementations were not commonly available in machine-learning frameworks. Furthermore, while there are musical difference operations that work on hierarchical data structures, such as those used by MusicDiff and Mupix (see Section 2.5.2), they are limited in scope; for example, MusicDiff primarily focuses on pitches, rests, and their timings, excluding other notational aspects. Moreover, generalized methods for computing differences between graphs, which involve finding the minimal sequence of operations to transform one graph into another (Delugach and Moor 2005), are computationally more challenging and less well-studied than the equivalent operations on sequences (see Section 2.4.3). Given these considerations, the use of graph structures for error analysis in symbolic music was deemed not sufficiently established for the purposes of this dissertation. However, this area holds potential for future research, further explored in Section 7.3.2.

Token-based representations, which use individual tokens to represent distinct categorical values, are a common tool in the field of NLP. In the context of music, token-based formats can be either semantic, representing precise pitches and musical instructions as interpreted by performers, or agnostic, which represent symbols as they appear on the page. Figure 3.5 provides an illustration of both agnostic and semantic encodings for the same musical passage, with further details on agnostic encodings discussed in Section 3.1.2. An instance of a token-based representation in music is seen in the study by Hawthorne et al. (2021), focused on Automatic Music Transcription (AMT). Although their representation does not encompass the full range of musical notation, it could be expanded to incorporate more comprehensive musical elements such as beams, articulation markings, ties, and other notational details. For the purposes of this research, an agnostic token-based representation is more advantageous than a semantic one. It aligns more closely with three key criteria outlined earlier: it offers higher granularity, has

a smaller vocabulary size, and is more suitable for representing the errors produced by OMR processes.

Agnostic representations, by detailing each symbol in a musical score as a separate token, offer a higher level of granularity compared to most semantic representations. For instance, a musical element like a dotted eighth note with an accidental would typically be denoted by a singular token in a semantic representation, containing information about its pitch and duration. However, in an agnostic format, the accidental, the note, and the dot are each distinct elements. This granularity allows an error detector to precisely identify which specific token within a musical element is erroneous.

This approach of separating elements into individual tokens also results in a reduction in the total number of unique tokens needed to represent a music corpus. The extent of this reduction depends on the specific semantic and agnostic encoding schemes used. Taking the semantic MusicXML specification as an extreme case, the `<note>` element may encapsulate multiple attributes or sub-elements, like duration, pitch, tied notes, dots, accidentals, stem direction, dynamics, articulations, and grace notes. A token-based equivalent scheme would necessitate a large vocabulary, as every possible combination of these musical elements would need a unique token. In the field of NLP, a strategy employed to manage large vocabularies is to break down words into smaller subword units. For example, the relatively rare word "suboptimally" can be segmented into more frequent subwords `[sub, optim, al, ly]`. This technique is analogous to the proposed method of using more granular agnostic tokens.

Finally, the types of errors made by OMR are more naturally represented using an agnostic representation. In a situation where an agnostic encoding identifies a missing accidental, a semantic representation might perceive it as a note pitch being swapped for another. Although both interpretations are valid, presenting the error as a "missing accidental" is likely more intuitive for the end user of the error detector, and more closely aligns with the type of operation that one would have to perform in a notation editor in order to correct the note.

5.2.3 Agnostic Encoding Scheme

In this section, I specify the agnostic encoding scheme I have developed for use as the input format for the error detector.

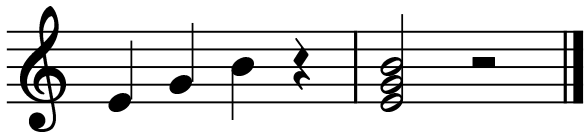
As discussed in Section 3.1.2, agnostic encodings in OMR systems typically serve as an intermediary stage prior to the final semantic output in formats like MusicXML. For optimal

performance, it is preferable to have the error detector work directly on the agnostically encoded output from an OMR process. This approach is advantageous because the conversion from agnostic to semantic output is often complex and it may conceal critical error-related information about the OMR process. For example, if an OMR system identifies a discrepancy between the total duration of the notes in a measure and the duration required by the measure's time signature, this signals a probable error in recognition. However, in the transformation to a semantic format, the system might modify the musical content (such as adding rests) to conform to the time signature. This alteration ensures the semantic output's validity and compatibility with other notation editors but at the same time masks the original error's nature. Therefore, by employing an agnostic encoding scheme directly, the error detector can access and analyze these raw, unaltered signals from the OMR process. This approach prevents the loss of error-indicative information that may occur during the agnostic-to-semantic conversion process.

To operate the error detector on this internal agnostic representation, however, I would be constrained to using OMR systems that not only employ a sequence-like agnostic representation but are also either open-source or provide access to this internal representation. Instead, to maintain the method's generality and applicability to any OMR system, I have chosen to treat OMR systems as black boxes. This approach assumes that only the semantically encoded output from the OMR process is accessible. Consequently, in order to operate on an agnostic encoding, I must translate semantically encoded files into an agnostic format as a first step.

In this section, I outline the specific scheme developed for translating semantically encoded music files into an agnostic encoding. This scheme is largely inspired by the one defined by Calvo-Zaragoza and Rizo (2018b), but it has been adapted to accommodate a broader spectrum of phenomena found in Common Western Music Notation (CWMN). The translation scheme is a critical component of the error detection process, as it ensures that the method can be universally applied across any OMR systems regardless of its underlying architecture.

To transform musical scores from a semantic format, such as MusicXML or humdrum **kern, into an agnostic format, I utilize the Python package `music21` (Cuthbert and Ariza 2010). The process begins with parsing the semantically encoded file into a `music21` Stream object. This object hierarchically represents a musical piece as a collection of parts. Each part corresponds to a staff line in the score and contains a sequence of measures. These measures, in turn, contain musical elements like notes, rests, and other markings. For multi-part music, this script processes each part individually and afterwards combines them together into a single sequence of tokens.



(a) A short musical excerpt.

```
clef.treble
quarter.noBeam.up.pos1
quarter.noBeam.up.pos3
quarter.noBeam.up.pos5
rest.quarter
barline.regular
half.noBeam.up.pos1
∧
half.noBeam.up.pos3
∧
half.noBeam.up.pos5
rest.half
barline.final
```

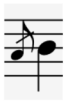
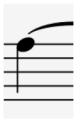
(b) The agnostic encoding of the excerpt.

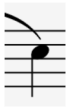
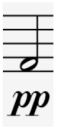
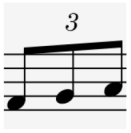
Figure 5.1: An example of how the caret ∧ token is used in the agnostic encoding scheme. Note how in chords the ∧ token is used between tokens in the same horizontal position.

For the agnostic encoding of a single staff in polyphonic music, I implement a procedural, deterministic set of rules to systematically represent musical symbols. A detailed description of how the agnostic encoding scheme treats each type of musical symbol, detailing unique edge cases and exceptions, can be found in Table 5.1. The process involves listing all symbols on the staff sequentially from left to right. In cases where two symbols occupy the same vertical space, such as notes in a chord or an articulation mark above a note, they are encoded in order from the bottom symbol to the top symbol. To differentiate between symbols that are adjacent horizontally and those that are adjacent vertically, I introduce a unique token containing a caret symbol ∧. This token is placed between tokens that are vertically adjacent. Additionally, tokens that hold semantic significance in their vertical positioning on the staff are paired with a `position` attribute. This attribute indicates the vertical location of the glyph relative to the staff lines. A note positioned on the lowest space of the staff is assigned a position of 0, while a note on the line immediately above this line is given a position of 1. This pattern continues upwards, with each subsequent line and space receiving a higher position value. See Figure 5.1 for a simple example of how the position of each note on the staff is encoded and how the ∧ icon is used to mark chords. This encoding strategy ensures that each symbol's spatial relationship to the staff, whether horizontal or vertical, is preserved in the agnostic format.

The fundamental set of musical symbols I consider is as follows: notes (including grace notes), rests, augmentation dots, clefs, time signatures, accidentals, articulations, dynamics, ties, slurs, barlines, and tuplet indicators. This categorization is broader than many OMR

Table 5.1: A table showing all possible glyph types in the agnostic encoding scheme, and how they are translated into tokens from their semantic representations.

Glyph	Example
Note `[duration].[beam].[stem].[position]` Notes in chords are listed from lowest to highest staff position, even when an interval of a 2nd would force one note to be engraved slightly to the left of another. Possible values for `beam` are `noBeam`, `start`, `continue`, `end`.	 `8th.noBeam.down.pos7`
Grace Note `[type].[duration].[beam].[stem].[position]` Treated the same as notes, but with an additional parameter that describes whether the note has an oblique slash through it or not. Possible values for `type` are `acciaccatura`, `appogiatura`.	 `acciaccatura.8th.nobeam.up.pos4`
Accidental `accid.[type].[position]` Accidentals assigned to notes in chords are listed from lowest to highest staff position, with ∧ tokens between them as if they were in a vertical stack, even when standard engraving rules would stagger the accidentals horizontally for visual clarity. Possible types are `doubleflat`, `flat`, `natural`, `sharp`, `doublesharp`.	 `accid.sharp.pos3`
Dot (duration) `dot.[position]` Placed at the same position as its corresponding note if the note lies on a space, or one position above if the note lies on a line. Duration dots assigned to notes in chords are listed from lowest to with ∧ tokens between them as if they were in a vertical stack.	 `dot.pos4`
Ties `[tie].[status].[position]` Ties are represented by two tokens: one where the tie begins, placed immediately after the first note, and one where the tie ends, placed immediately before the second note. Possible statuses are `begin` and `end`. Multiple chains of tied notes are marked by multiple pairs of `begin` and `end` ties.	 `tie.start.pos9`

Glyph	Example
Slurs `[slur].[status].[position]` Slurs are represented in the same way as ties, with a token placed immediately after the first note or chord and another token immediately before the second note.	 `slur.end.pos6`
Rest `rest.[duration]` As `music21` does not detect the vertical positions of rests when parsing MusicXML files, this encoding method does not assign rests a position on the staff, and assumes them to be in the middle of the staff.	 `rest.16th`
Articulation `artic.[type].[position]` Possible articulation types are all those defined by `music21`, including accents, tenuto, staccato, staccatissimo, bowing markers, harmonics, pizzicato, and others that do not appear in the datasets used in this research. Articulations that can be either above or below the note are assumed to be below the note when the stem points upwards or is absent, and above the note otherwise. The position is, for most types of articulation, assumed to be the closest staff space lower or higher than the note head, depending on stem direction. For articulation marks that always appear above the staff, they are assumed to be placed at position 10 (the space above the top staff line).	 `artic.accent.pos0`
Dynamics `dynamics.[type]` Dynamics are assumed to be under all other notes and glyphs at the horizontal position where they appear. Possible dynamic marking types are all those defined by `music21`, using their short name (e.g., `mf`, `pp`, `fff`).	 `dynamics.pp`
Tuplet Indicator `tuplet.[division]` Tuplet indicators are assumed to be above or below the center note depending on if the stem direction of the notes is up or down. Since the precise position of the tuplet indicator depends on the surrounding musical material, they are not given a position in this encoding. Tuplets written as ratios (e.g., 3:2, 4:5) are not handled.	 `tuplet.3`

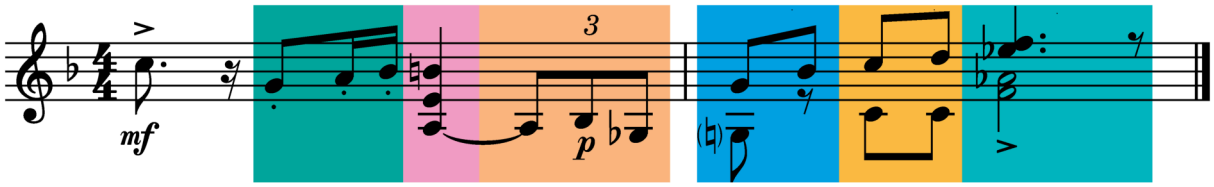
methods, which describe noteheads, stems, beams, and flags as distinct shapes. My approach aims at translating already-encoded semantic music files, which inherently assume the correct placement of note stems and that beams are connected to the appropriate notes. Therefore, treating these aspects of CWMN as distinct tokens would contribute no additional information to the agnostic encoding.

In most cases the semantic-to-agnostic conversion procedure is unambiguous. However, specific situations demand that I work out how a semantically encoded music section *would* be represented in an engraving to decide the precise sequence of agnostic tokens. These can be scenarios where incorporating extensive automatic engraving logic into the encoder would be necessary for total consistency. In these instances, I prefer to underspecify the relative positions of elements on the page, permitting the encoding to retain a level of imprecision. I prioritize simplicity and consistency, opting for representations that may not correspond to an aesthetically appealing score and instead leaning towards choices that minimize the overall vocabulary size of possible tokens.

After converting each staff from a semantic music file into an agnostic representation, these sequences need to be collected into a single sequence suitable for input into a machine-learning model. My approach concatenates each measure of every system in the score according to their visual sequence on the page. To start, the first measure of the first part is presented, followed by the first measure of the second part, continuing this pattern until the first measures of every part in the score are concatenated together. Then, a unique `backToTop` token is appended before the sequence continues: concatenating the second measure of the first part, the second measure of the second part, and so on.

This method of interleaving the parts by measure is based on the assumption that the model would benefit from information on vertical harmony and polyphonic texture. Consequently, it would be advantageous for events that are temporally close to also be proximal in the sequence of tokens. While other strategies to unify multiple polyphonic staves into a singular one-dimensional token sequence exist (e.g., interleaving based on events within each beat), a denser interweaving of system staves results in errors confined to a single staff becoming dispersed, making them less precisely represented when processed with an ANW alignment.

In Figure 5.2, I provide an example of a two-measure music snippet and its corresponding agnostic encoding, created specifically to demonstrate various glyphs and edge cases in the conversion algorithm. Key phenomena in both the score and the agnostic encoding are highlighted



(a)

```
clef.treble
accid.flat.pos4
timeSig.4/4
dynamics.mf
^
eighth.noBeam.down.pos5
^
articulation.accent.pos4
duration.dot.pos6
rest.16th
articulation.staccato.pos2
^
eighth.start.up.pos2
articulation.staccato.pos2
^
16th.continue.up.pos3
articulation.staccato.pos4
^
16th.stop.up.pos4
accid.natural.pos4
quarter.noBeam.up.pos-4
^
quarter.noBeam.up.pos0
^
quarter.noBeam.up.pos4
tie.start.pos-4
tie.end.pos-4
eighth.start.up.pos-4
dynamics.p
^
eighth.continue.up.pos-3
^
tuplet.3
accid.flat.pos-5
eighth.stop.up.pos-5
barline.regular
accid.natural.pos-5
eighth.noBeam.down.pos-5
^
eighth.start.up.pos2
rest.eighth
^
eighth.stop.up.pos4
eighth.start.down.pos-2
^
eighth.start.up.pos5
eighth.stop.down.pos-2
^
eighth.stop.up.pos6
accid.flat.pos3
^
accid.flat.pos7
half.noBeam.down.pos1
^
half.noBeam.down.pos3
^
articulation.accent.pos4
quarter.noBeam.up.pos7
^
quarter.noBeam.up.pos8
duration.dot.pos8
^
duration.dot.pos8
rest.eighth
barline.final
```

Figure 5.2: (a) An engraved two-measure snippet of music. (b): The agnostic encoding of the above snippet, using the scheme defined in Table 5.1. Highlighted regions of the agnostic encoding correspond to highlighted regions of the same color in the musical snippet, with annotations corresponding to some of the highlighted regions.

with specific colors. I present detailed explanations for each highlighted section below:

- Green Highlight: Barred notes with a `continue` modifier do not distinguish between adding a bar to transition to a shorter note duration (as in this example), removing a bar, or maintaining the same note duration. The number of bars can be inferred from the specified durations of the notes involved.

- Pink Highlight: For chords with tied notes, the `tie.start` token appears in the agnostic sequence after every note in the chord that is being tied (or before every note, if the tie is ending).

- Peach Highlight: In a fully implemented engraving system, the exact placement of a tuplet indicator over a series of tuplets varies depending on the surrounding musical context. I generally place it above the middle note of the tuplet, but do not include its precise position as with note heads.

- Blue Highlight: Courtesy accidentals, like the natural sign here, are included in the agnostic representation only if they are explicitly marked in the semantically encoded input. The position of the eighth rest in this engraving is not on the central line of the staff due to the presence of two simultaneous, rhythmically independent voices, a detail not reflected in the agnostic representation.

- Orange Highlight: In cases where a single staff has multiple voices with barred notes, this agnostic scheme can create ambiguity regarding which notes are barred together. While context may make the proper reading unambiguous in simpler passages, more complex sections might present challenges.

- Blue-Green Highlight: Notes played simultaneously are treated as having the same horizontal position, even if they are offset in the engraving. For instance, the dotted-quarter F is notated as if it is directly above the E♭, even though it is engraved slightly to the side. Articulation marks, like the one written below the two half notes but appearing above them in the agnostic encoding, are positioned based on the note stem direction. I assume all articulation marks to be above the note for notes with downward stems and below the note for upward stems.

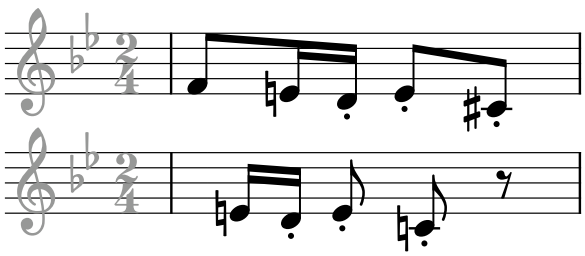
The intent of this system is to be able to represent all musical pieces in CWMN, but the described encoding mechanism does not guarantee a distinct agnostic representation for every score. In instances where a single staff indicates multiple voices with varying rhythmic structures, like those observed in orchestral compositions or intricate piano pieces, the scheme as presented fails to differentiate notes from one voice and notes in another. This limitation is evident in the second measure of the musical excerpt in Figure 5.2. Here there are two groups of two eighth notes beamed together, but the encoding does not explicitly note which eighth notes beam to which other ones. In this particular case, the correct reading is obvious from context, as the notes are distant in pitch, but if these notes were close in range the ambiguity would not always be resolvable. Consequently, an accurate recreation of the original semantic score from its agnostic counterpart may not always be feasible. However, since the agnostic encoding step serves only as a specialized data preprocessing step to convert the data into a form appropriate for machine learning, it is not necessary that it be perfectly reversible or faithful to the original score; all that matters is that it retain enough musical information that the model can learn from it.

The goal of this error detector system is not just to make predictions on agnostically encoded sequences, but to annotate a semantically encoded file with the predicted locations of errors, rendering the results human-readable. Converting from an agnostic representation back into a semantic representation is a nontrivial task and an area of research in itself (see Section 3.1.2), so I avoid this. Instead, when converting from a semantic to an agnostic format, I retain additional indices that note the part, measure, `music21` object, and chord position from which each agnostic token is derived. I refer to this as the *agnostic-to-semantic alignment* of the score. After inference with the error detector, each token's prediction can be cross-referenced back to a unique object or object attribute in the `music21` stream, and from there referenced back to the original semantically encoded score. This allows the identification of which element of the score corresponds to which agnostic token when highlighting errors.

5.2.4 Defining Errors with the Affine Needleman-Wunsch Aligment

Given two passages of music in agnostic format as defined by my encoding scheme, I define the error between them using the ANW alignment algorithm, which itself is described in detail in Section 2.4.3. The ANW alignment produces an *alignment record* showing what kinds of operations, one would need to perform on one sequence to transform it into another, where an operation is either a deletion, an insertion, or a replacement. The error detector operates by predicting, based on OMR output that needs to be corrected, a simplified version of this record of necessary operations. In effect, it answers the question: *Given a musical score that contains errors, at which positions would you need to perform operations to correct those errors, according to an ANW alignment?*

Figure 5.3 illustrates an ANW alignment between two music excerpts in agnostic format, along with the corresponding record of operations. Note that the length of this operation record does not match that of either musical input sequence. Its length is always equal to the length of the shorter of the two sequences plus the number of insertions and deletions necessary to turn one into the other. Training a machine-learning model to replicate a variable-length operation record would resemble more the task of sequence-to-sequence translation rather than error detection, which is feasible. However, it would no longer produce a binary classification on individual input tokens, and instead give a list of operations. The problem here is that the association between operations and sequence positions is dependent on the model having predicted the correct number of operations. For example, consider the sequence of operations in the third



(a) A correct score (top) and version with OMR errors created by PhotoScore (bottom).

Original	OMR	Operation
barline.regular	barline.regular	SAME
8th.up.start.pos2	_	INSERT
accid.pos1.natural	accid.pos1.natural	SAME
16th.up.continue.pos1	16th.up.start.pos1	REPLACE
dot.pos-2	dot.pos-2	SAME
∧	∧	SAME
16th.up.end.pos0	16th.pos0.end	SAME
dot.pos-1	dot.pos-1	SAME
∧	∧	SAME
8th.up.start.pos1	8th.up.noBeam.pos1	REPLACE
accid.pos-1.sharp	accid.pos-1.natural	REPLACE
dot.pos-3	dot.pos-3	SAME
_	∧	REPLACE
8th.up.end.pos-1	8th.up.noBeam.pos-1	REPLACE
_	8th.rest	DELETE
barline.regular	barline.regular	SAME

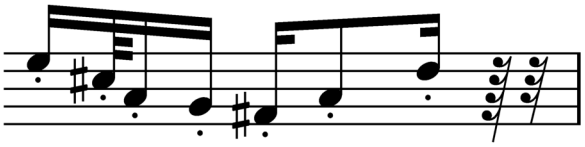
(b) The Needleman-Wunsch alignment between the agnostic encodings of the two measures. The **Operation** column shows the operations needed to change one column into the other.

Figure 5.3: Mendelssohn's Quartet No. 1 in E♭ Major Op. 12, Mvt. 2, measure 10, 2nd violin part. Two measures are shown, one with OMR errors, along with the result of their ANW alignment.

column of the table in Figure 5.3. What if, instead of predicting five `SAME` operations in a row in the middle of this excerpt, the model had predicted six? In that case, the subsequent `REPLACE` and `DELETE` operations would be shifted over by one space, and there is no way to deduce from the model's predictions alone which tokens it originally meant to replace or delete. For this reason, I abandoned the notion of directly predicting the record of operations.

To adhere to the task of binary sequence classification, I must adjust the operation record such that each token has exactly one corresponding operation. I do this by removing all runs of insertions (that is, places where one would need to insert a token in order to correct the OMR output) and marking the token immediately before the gap as an error instead. This discards information not only on what kinds of operations must be done at which points, but all information on how many tokens must be *inserted* to correct a score, when an insertion must be made. Figure 5.4 shows how this method marks tokens as erroneous in a measure of music.

(a) The measure in raster image format, as distributed with the MSQD.



(b) The measure after being processed by PhotoScore's OMR process.

Token	Error Here?	Current Note
`barline.regular`	-	-
`articulation.staccato.pos6`	✓	E5, 16^{th}
∧	-	E5, 16^{th}
`16th.up.start.pos8`	✓	E5, 16^{th}
`accid.sharp.pos6`	✓	C♯5, 64^{th}
`articulation.staccato.pos4`	✓	C♯5, 64^{th}
∧	-	C♯5, 64^{th}
`64th.up.continue.pos6`	✓	C♯5, 64^{th}
`articulation.staccato.pos2`	✓	A4, 16^{th}
∧	-	A4, 16^{th}
`16th.up.continue.pos4`	✓	A4, 16^{th}
`articulation.staccato.pos0`	✓	G4, 16^{th}
∧	✓	G4, 16^{th}
`16th.up.stop.pos3`	✓	G4, 16^{th}
`accid.sharp.pos2`	✓	F♯4, 16^{th}
`articulation.staccato.pos0`	-	F♯4, 16^{th}
∧	-	F♯4, 16^{th}
`16th.up.start.pos2`	-	F♯4, 16^{th}
`articulation.staccato.pos2`	-	A4, 8^{th}
∧	-	A4, 8^{th}
`eighth.up.continue.pos4`	✓	A4, 8^{th}
`articulation.staccato.pos4`	-	D5, 16^{th}
∧	-	D5, 16^{th}
`16th.up.end.pos7`	✓	D5, 16^{th}
`rest.64th`	✓	-
`rest.32nd`	✓	-
`barline.regular`	-	-

(c) The version of the measure in (b), encoded agnostically and annotated with the locations of errors as defined by the ANW alignment method. The third column shows which note in the OMR measure is referred to by the current agnostic token.

Figure 5.4: Mendelssohn's String Quartet no. 3 in D Major, Op. 44 No. 1., mm. 2, second violin part. This figure shows a scan of the measure, the results of an OMR process on that scan, and the positions where errors lie according to the scheme defined in this section.

5.3 Dataset Composition

In this section, I detail the datasets for the experiments in Chapter 6 to assess the performance of the error detector. The ideal dataset for training an error detector to identify OMR errors would consist of a large collection of musical scores in symbolic format with OMR mistakes. These scores would be paired with their corrected versions and the images used to generate the symbolic files through the OMR process. With such a dataset, a supervised training approach would be straightforward. I would perform an ANW alignment between a corrected score and its erroneous OMR counterpart, deduce the error positions from the operations record as outlined in Section 5.2.4, and then train a model with the erroneous score as input and the deduced errors as the binary targets for prediction.

However, a dataset of this size and specificity does not exist. Producing it would demand considerable effort and time. Automating OMR processes at the scale required for deep learning, using current commercial tools, is challenging as the workflow of all existing tools assumes an amount of human interaction in the process to check for errors. Additionally, I assume that each music genre will necessitate a distinct error detector model, so an ideal approach would permit training with a limited database of OMR errors. Therefore, I seek to develop a method that leverages a modestly sized dataset of natural OMR errors and their corresponding corrected scores. I refer to errors artificially integrated into correct scores for training data creation as *synthetic* errors, while errors stemming from an OMR process on an actual score image are termed *natural* errors. Training occurs first on synthetic errors introduced into a larger dataset of error-free scores, followed by a fine-tuning step that trains on the small dataset of natural errors.

5.3.1 Dataset of OMR Errors

The datasets of natural OMR errors used for training the error detector are the Mendelssohn String Quartet Dataset (MSQD), assembled by deGroot-Maggetti et al. (2020) and the OpenScore String Quartets (OSSQ) corpus, assembled by Gotham et al. (2023). These sources provide a wide range of OMR errors for analysis and model training.

The MSQD was specifically compiled for a case study focused on correcting scores post-OMR processing (deGroot-Maggetti et al. 2020). It includes six of Mendelssohn's string quartets, totaling 24 individual movements. Each movement is available in four distinct versions: the initial

output from PhotoScore's OMR process (refer to Section 3.3) and three subsequent versions, each revised and corrected by different human reviewers under increasingly strict correction criteria. The fourth and final version is considered entirely accurate. The MSQD is unique in that it provides both corrected score versions and their corresponding OMR-processed counterparts. Additionally, the dataset enables evaluation of the error detector's capability to identify errors commonly overlooked by humans during initial correction passes. Figure 5.4 shows an excerpt from one of these quartets, showcasing the original scanned image used in PhotoScore, the resulting raw MusicXML output, and the error locations as determined by the error scheme defined in Section 5.2.4.

The OSSQ project is an initiative that seeks contributions from skilled transcribers to symbolically encode historically significant string quartets that currently lack public-domain symbolic encodings (Gotham et al. 2023). These quartets are transcribed from public-domain scans available on International Music Score Library Project (IMSLP), with each score transcribed and then reviewed by at least two distinct musical experts. The scores transcribed come from a variety of sources, including low-resolution flatbed scans of modern printed editions, high-resolution photographs of historical printed sources, and some digitally engraved Portable Document Format (PDF) files. Scores in this dataset are not paired with versions containing OMR errors, but since each score is associated with a publicly available scan, it is straightforward to create this data. For each fully transcribed and reviewed quartet within the OSSQ, I retrieve the corresponding score scans from IMSLP and process them using PhotoScore to produce new MusicXML files containing natural OMR errors. The only interaction I have with the process is within the PhotoScore correction interface, where I ensure the correct identification of the number of staves on each page and that the staves are grouped correctly into systems of four parts.

Several string quartets transcribed by the OSSQ project are not suitable for inclusion in this dataset. Some quartets, being 20th-century atonal pieces, are excluded due to concerns about providing the model with highly unpredictable input. Others, transcribed from handwritten score scans, are also excluded since the handwritten score recognition engine in PhotoScore differs from the one for printed scores; for dataset uniformity, it would be preferable to have a consistent source of natural OMR errors, since OMR on handwritten scores is a much more difficult problem. Finally, some particularly degraded scans caused PhotoScore to crash or to output a score too corrupted to be read by other programs, and I discard those as well.

From the OSSQ I was able to collect 46 additional string quartets and versions of them containing natural errors. Most of these string quartets are four-movement pieces, but a substantial minority, mostly written by Joseph Haydn, are shorter one-movement pieces. Combined with the existing six quartets from the MSQD, this dataset contains a total of 52 pieces containing natural errors.

5.3.2 Synthetic Dataset

I now outline the composition of the large dataset containing correct scores into which I will introduce synthetic OMR errors. I refer to this dataset as the *synthetic dataset*. It would be ideal for this dataset to be as consistent as possible in genre, focusing specifically on a single era of music to match the contents of the MSQD and the scores from the OSSQ. However, there only exist so many Classical and Romantic string quartets, and even fewer for which high-quality symbolic transcriptions are freely available. Consequently, I compile a dataset comprising string quartets and other four-part scores from the 19th century and earlier, excluding contemporary genres where the concept of an "obviously wrong note" is less clear. The most modern piece included is Claude Debussy's String Quartet in G Minor, initially published in 1893. While I could have incorporated music from the same period and style but with diverse ensembles, I prioritize consistency in instrumentation. Notably, solo piano scores typically have a higher note density than string quartet staves, suggesting that the nature of OMR errors in solo piano scores would notably vary.

The synthetic dataset I have assembled is composed of several existing datasets:

- **Kernscores-Quartets**: Every symbolic file in Kernscores[2] that is a string quartet. This comprises the complete string quartets of Haydn, Mozart, and Schubert.
- **The Annotated Beethoven Corpus (ABC)**: A corpus containing all of Beethoven's string quartets (Neuwirth et al. 2018). The accompanying annotations, for which the corpus is named, are not used.
- **The subset of the OSSQ**: I mention above that not every string quartet transcribed by the OSSQ was suitable for inclusion in the small dataset containing natural errors. If I was unable to retrieve a workable score from running PhotoScore on a given piece's public domain scans, I add its human-corrected version to this dataset instead.

2. `http://kern.ccarh.org/`

Table 5.3: A table showing the composition of the dataset assembled for training the error detector. Here ABC stands for the Annotated Beethoven Corpus and OSSQ is the OpenScore String Quartets corpus.

Dataset	**# symbolic files**	**# tokens**	**# tokens after transpositions**
Kernscores	361	1,270,746	3,812,240
Musescore	210	1,185,855	5,637,895
ABC	70	393,125	1,179,375
Subset of OSSQ	9	190,248	570,745
Total	650	3,554,933	17,034,095

- **Musescore-Quartets**: Musescore is, in addition to being a popular free music notation application, a website[3] where users are invited to upload original music, transcriptions, and arrangements for download and comment by other users. I downloaded every file I was able to access on the MuseScore service that was a string quartet and went through each of them to manually exclude all those that were, in terms of style, too modern for a model to learn anything from it applicable to Classical or Romantic patterns of tonality and structure.

To further increase the size of the dataset, I include two transpositions of each file: a version transposed a second up, and another transposed a second down. I restrict the transpositions to these two in order to keep the transposed pieces mostly within the range of the instruments of a string quartet. These transpositions are performed by `music21`'s built-in transposing functionality, and so the pieces are not re-engraved by a notation editor in the process. Table 5.3 shows a breakdown of the number of tokens in the agnostic encodings of each segment of the dataset, as well as the number of tokens in the datasets after augmentation by transposition.

5.3.3 Data Augmentation

This section details the data augmentation techniques employed to generate additional training data for the error detector by introducing errors into the large dataset of human-corrected string quartets. These techniques draw inspiration from comparable data augmentation methods common in GEC and GED domains, as detailed in Section 2.2.1.

The primary objective is to emulate errors similar to those that PhotoScore might introduce, ensuring that they are similar to those observed in the smaller dataset containing natural errors. The dilemma faced here, though, is that the OMR errors are themselves complicated enough phenomena that accurately modeling them would require machine-learning methods. This im-

3. `https://musescore.com/`

plies that a portion of the datasets containing OMR errors would need to be reserved to train this error emulation model. Additionally, this segment of the dataset must be isolated from the test data to prevent the model from encountering information from identical pieces during both its training and testing phases, as a matter of data hygiene. In contexts like GEC and GED, such error generators use sequence-to-sequence models, which function in a manner akin to automatic language translation systems. This approach is not viable here, given the limited size of the OMR dataset. A model that is powerful enough to be capable of recreating those errors would overfit if only given a small amount of training data. In effect, the technique I have proposed to ameliorate the data scarcity problem itself faces a data scarcity problem.

Another reason that I do not pursue a more complicated sequence-to-sequence model is that I plan to generate augmented data on the fly. This will add synthetic errors to each individual batch of agnostic sequences during training, which would allow for an effectively unbounded amount of augmented data. It is important that the augmentation method be fast enough to not cause a bottleneck in the training loop. When generating sequences, sequence-to-sequence models (including the Recurrent Neural Network (RNN) and the Long Short-Term Memory (LSTM); see Section 4.2.4) require a number of evaluations proportional to the length of the sequence itself. Given the volume of training data needed, using a sequence-to-sequence model to generate errors as part of the training loop would slow down the training process.

In practice, though sophisticated error generation methods do tend to yield better training performance in other fields, it is not necessary that a deep learning model for error detection or correction be trained on errors that *exactly* match the characteristics of the errors one is trying to correct in all respects (See Section 2.2.1 for discussion of this point in the context of GEC and GED). In the subsections below I describe three data augmentation procedures that aim to strike a balance between generating synthetic errors that closely resemble natural OMR errors, efficiency in using only small amounts of data for analysis, and simplicity in model design so that they do not overfit. The impact of each data augmentation method on the performance of the resultant model is separately assessed in Chapter 6.

5.3.3.1 Simple Data Augmentation

The canonical method of generating synthetic errors is to introduce random noise into data with some intensity. This method does not require using a portion of the dataset for error analysis. For each token in an agnostic music sequence, I independently decide whether or not that token

will be modified given a modification probability between 0 and 1. Then, for all tokens marked with modification, perform one of three possible operations with equal probability:

- `INSERT` another token into the sequence one space ahead, randomly chosen from the vocabulary of agnostic tokens based on its overall prevalence in the training data.
- `REPLACE` the token with some other token, randomly chosen from the vocabulary of agnostic tokens based on its overall prevalence in the training data.
- `DELETE` the token.

Since this method chooses randomly from the full vocabulary of tokens, it is possible for the generated agnostic sequence of tokens with errors to be musically impossible, in that it does not correspond to any possible engraving of music. It could generate phenomena like clefs in the middle of chords, notes with beams that start and never end, and accent marks applied to glyphs that are not notes. I hypothesize that a machine-learning method trained on this kind of data could perform relatively well just by identifying things that are impossible to engrave, and a model trained in such a way may not extrapolate well to subtler kinds of errors. This motivates the design of more sophisticated methods of generating synthetic errors.

5.3.3.2 Probability-Based Data Augmentation

To enhance the realism of synthetic OMR-like errors introduced into sequences of agnostic tokens, it is necessary to acquire statistical data on the typical errors made by PhotoScore's OMR system, particularly in the context of string quartets. For this purpose, I allocated a small segment of the natural OMR errors dataset for detailed error analysis. This subset consists of eight pieces, including only individual movements and single-movement works, selected to represent a range of OMR output qualities. For each piece, an ANW alignment is performed between the OMR output and its manually corrected version. This alignment process enables the extraction of statistics on how different types of musical symbols are typically affected by the OMR process. Table 5.4 illustrates the typical outcomes for quarter rests and quarter notes on the top line of the staff in scores processed by PhotoScore.

As one might anticipate, these symbols most often remain unchanged (indicated by the `SAME` operation, signifying no error by the OMR process), with deletion (`DELETE` operation) being the most frequent error. These gathered statistics form a probability distribution from which

Table 5.4: An example probability distribution analyzed from the OMR data, indicating the types of operations that the OMR process most commonly performs on quarter rests (left) and quarter notes on the 10th position of the staff, sitting just above the top line with stem pointing down (right). The right column (**P**) of each table notes the probability of each operation. Only the first few most common operations are shown for each.

Operations on `quarter.rest`

Operation	**P**
`SAME`	0.627
`DELETE`	0.142
`INSERT 8th.rest`	0.032
`INSERT barline.regular`	0.032
`REPLACE 8th.rest`	0.013
`REPLACE` $\wedge$	0.009
`INSERT quarter.rest`	0.007
`REPLACE whole.rest`	0.003
`INSERT quarter.pos5.noBeam`	0.003
...	

Operations on `quarter.noBeam.down.pos10`

Operation	**P**
`SAME`	0.776
`DELETE`	0.042
`INSERT` $\wedge$	0.041
`INSERT rest.quarter`	0.012
`REPLACE rest.quarter`	0.012
`INSERT articulation.accent.pos10`	0.010
`REPLACE rest.eighth`	0.009
`INSERT rest.eighth`	0.008
`REPLACE duration.dot.pos10`	0.006
...	

I sample for each symbol in an agnostically encoded piece. Based on the sampled outcome, the corresponding operation is applied to each symbol. If the `SAME` operation is selected, the symbol remains unaltered. The probability distributions have long, thin tails with the majority of the probability mass concentrated on a few common errors. Consequently, this method is less prone to generating musically implausible agnostic sequences than simpler data augmentation techniques. Importantly, a symbol can only be replaced by another symbol if such a substitution has been observed at least once in the OMR error analysis dataset. This constraint ensures that the augmented errors remain grounded in realistic OMR error patterns.

5.3.3.3 Probability-based Data Augmentation with Heuristics

Observations of OMR output from PhotoScore reveal a noticeable tendency for errors to cluster or "clump" together, rather than being evenly distributed throughout a piece. Factors such as smudges on the original page or densely notated sections often lead to a series of contiguous errors, whereas cleaner and sparser sections may be error-free for extended stretches. The probability-based data augmentation method previously described does not account for this phenomenon, as it treats each token independently. In my preliminary experiments using this augmentation approach, I observed that the resulting models were significantly more adept at identifying isolated errors rather than sequences of consecutive errors. I hypothesize that this discrepancy arose because the models were seldom exposed to prolonged error sequences during training.

Instead of using a more complex error generation model, I use a heuristic method to force

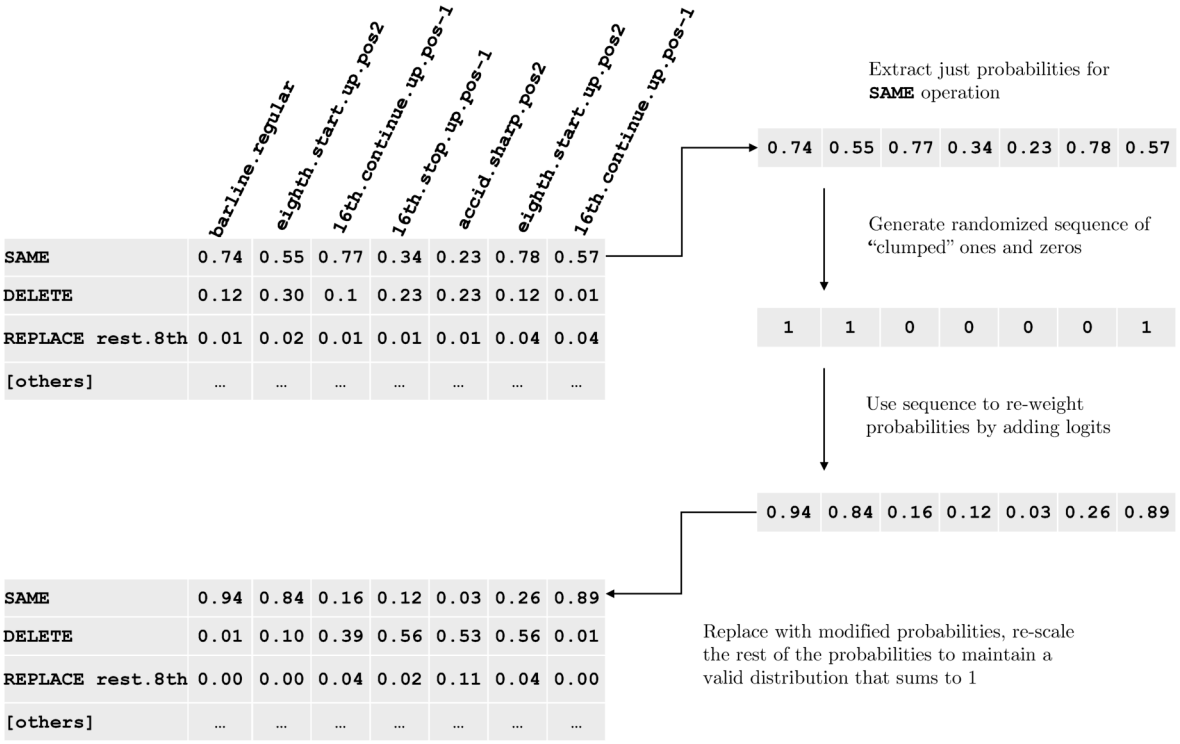


Figure 5.5: A diagram illustrating the heuristic method of generating errors in agnostic musical sequences that "clump" together into contiguous runs.

groups of adjacent tokens to be errors, outlined by Figure 5.5. First, given a sequence of agnostic tokens to introduce errors into, I generate a *clumping sequence* of ones and zeros that occur together in contiguous runs, in roughly the same way that errors occur in the OMR'ed scores. Then, I use this sequence of zeros and ones to modify the the error distributions (Table 5.4) such that ones increase the relative weight of the `SAME` operation in the distribution, and zeros decrease its relative share. This is described by the following formula, which is applied to each sequence element individually:

$$\mathrm{P}_{\text{final}}(\texttt{SAME}) = \sigma\left(\sigma^{-1}\left(\frac{1-s}{2} + s \cdot p_{\text{clump}}\right) + \sigma^{-1}(\mathrm{P}_{\text{original}}(\texttt{SAME}))\right) \tag{5.1}$$

where σ is the sigmoid logistic function and σ^{-1} is its inverse (sometimes called the *logit function*; see Section 4.2.1). The effect of the clumping sequence is the term p_{clump}, which takes on a value of 0 or 1 for every term in the sequence. s here is a parameter in the range $[0, 1)$ that represents the strength of this heuristic modification; when equal to 1, then the clumping sequence totally determines where errors are, and when equal to 0 it has no effect. Then I scale the remainder of each probability distribution so that they all still sum to 1 and are valid distributions. Then, they can be sampled to produce errors as in the previous section.

I created this heuristic to have as few modifiable parameters as possible, to ensure that it would not overfit to the types of errors that are in the data. Instead of tuning it to match actual metrics of the data, I manually changed the values of s and the average length of runs in the algorithm for clumping until the errors generated were similar to the ones in the OMR dataset by visual inspection.

5.4 Machine-Learning Architecture

I have chosen the Long Short-Term Universal Transformer (LSTUT) as the primary machine-learning architecture for the error detection task. The detailed specifications of the LSTUT and the rationale behind selecting it over other architectures are thoroughly discussed in Section 4.3.5. The LSTUT is a Universal Transformer bookended by two bidirectional LSTM layers, as detailed in Section 4.2.5. The core of this architecture, the Universal Transformer, is a variant of the vanilla Transformer. It consists of stacked multi-head self-attention layers, with the distinct feature that all these layers share the same weights. Conceptually, the Universal Transformer can be understood as a single multi-head self-attention layer that processes the input repeatedly over multiple iterations.

In implementing the network, instead of using the full Query-Key-Value (QKV) self-attention mechanism, which requires $O(n^2)$ memory relative to the input sequence length, I opt for a kernel-based linear attention mechanism. This approach is described in Section 4.3.4.3. The specific version of linear attention utilized in this study is sourced from the `fast-transformers` Python package[4], an extension to the PyTorch machine-learning framework (Paszke et al. 2017).

The full set of architectural parameters that define an instance of the LSTUT is as follows:

- The input sequence length n. When using full $O(n^2)$ self-attention, sequence length is not a required architectural specification; with the linear attention implementation I use, however, it must be specified ahead of time.
- The size of the token embedding e that the model operates on.
- The number of Bidirectional LSTM Layers l in the pre- and post-Universal Transformer LSTMs.
- The number of heads h in each self-attention layer.

4. `https://fast-transformers.github.io/`

- The hidden dimension d of each self-attention mechanism (i.e., the size of the Queries and Keys that each embedded token is projected to before linear dot-product attention is performed).
- The size f of the fully connected layers in each self-attention layer.
- The recurrence depth r of the linear multi-head attention step. This controls how many times the chain of self-attention layers will be applied to the input before moving on to the final LSTM.

For training the model, I employ the Adam optimizer, adhering to the default settings provided by its PyTorch implementation. To speed up the convergence of the training process, a cyclic learning rate scheduler is utilized (as outlined in Section 4.2.3). This scheduler alternates the learning rate between a specified minimum and maximum rate across every 32 training epochs. After each cycle, the maximum learning rate is halved, resulting in a gradual reduction in learning rate over time. Most aspects of the training procedure rely on default configurations set by PyTorch or the `fast-transformers` package. The kernel function for computing linear self-attention is left as its default formula: $\phi(x) = \text{ELU}(x) + 1$, where ELU is the Exponential Linear Unit (detailed in Section 4.2.1). Similarly, the Gaussian Error Linear Unit (GELU) is consistently used as the activation function in both the self-attention and LSTM layers.

The chosen loss function for training is the binary cross-entropy loss (referenced in Equation 4.15). This function is weighted to favor the identification of positive (erroneous) sequence elements, aligning the weighting with the proportion of erroneous tokens present in the OMR dataset. Although the Matthews Correlation Coefficient (discussed in Section 4.1.1) was considered as an alternative loss function, due to its potential suitability for high-recall tasks, it was found that training with this metric required more memory and exhibited less reliable convergence patterns compared to the binary cross-entropy approach.

5.4.1 Architectural Hyperparameter Search

The LSTUT is a specialized architecture in machine learning, designed specifically for analyzing the long-term dependencies of polyphonic symbolic music. Due to its niche application, there is no established "standard" configuration for the LSTUT in the broader machine-learning literature. Given the significant resources required to train deep learning models, including those with efficiency enhancements like kernel-based linear attention, it is impractical to exhaustively test

Table 5.5: A set of architectural parameters that define the Modified LSTUT, the values for each of them that are searched through in the randomized parameter search, and the value for each that was found to be optimal.

Description	Parameter	Values Searched	Optimal Value
Sequence Length	n	{64, 128, ... 2048}	512
Embedding size	e	{64, 128, ... 2048}	256
LSTM layers	l	{1, 2, 3}	1
Attention Heads	h	{4, 5, ... 14 }	12
Hidden dimension	d	{32, 64, ... 1024}	256
Fully Connected Dimension	f	{64, 128, ... 2048}	2048
Recurrence depth	r	{1, 2 ... 6}	4

every possible combination of hyperparameters for the LSTUT. To circumvent this challenge, I conducted a series of shorter training runs with partially randomized hyperparameters. The primary objective of these preliminary runs was not to identify an absolutely optimal architecture but to ascertain a reasonable range for each hyperparameter. This strategy is instrumental in understanding the relative significance of each parameter to the overall performance of the model. The insights gleaned from these initial experiments shape the design of the more comprehensive experiments presented in Chapter 6. By determining a viable set of hyperparameter values through these preliminary tests, the subsequent experiments can be more focused and efficient.

I ran these preliminary tests using the same setup that I will use in the main experiments, with the only difference being that every trial is restricted to run for no more than 90 minutes. This setup is described in detail in Section 6.1.1. Runs were conducted on a compute node containing four 12GB Graphics Processing Units (GPUs), for a total of 48 GB of memory. Changes in each of these architectural variables have a significant effect on the the memory usage and training time of the model. I filtered out architectures that would cause the training process to run out of memory (e.g., architectures with high values for more than one of d and r), which also enforces a trade-off between parameters that are costly to increase. I also filter out those parameter sets that would significantly underutilize the total GPU memory available. The models in these runs contained between three and six million trainable parameters each.

I conducted 48 trials, utilizing randomly chosen values for each parameter. Each potential architecture was initially trained to detect synthetic errors added to agnostic music sequences through the data augmentation method outlined in Section 5.3.3.3. Once the model converged on this task, it was fine-tuned on a collection of real OMR errors from the dataset described in Section 5.3. A predetermined training duration was set for each trial, after which they were

assessed using a test set of reserved data containing real OMR errors. Due to the short training time, I did not use a cyclic learning rate scheme (Section 4.2.3), as for some runs there would not be enough total epochs for a full learning rate cycle to pass. For each parameter, I identified the specific value that resulted in the minimal average validation loss across all the trials it was a part of. This value was then presumed to be the ideal setting for the corresponding parameter. Table 5.5 presents the outcomes of this architectural exploration, alongside the sets of potential values for every parameter that underwent scrutiny.

This process is grounded in the assumption that every parameter exerts a distinct, linearly separable influence on the network's operation. This assumption was made to avoid the need to examine every conceivable combination of parameters. Although this does oversimplify the effects of hyperparameters, it is an assumption that aligns with both theoretical frameworks and practical observations. Empirical studies centered on deep-learning LSTM tasks have found that hyperparameters largely operate independently of one another in terms of their effect on model performance (Greff et al. 2017). Additionally, such assumptions are commonly used when performing hyperparameter explorations on large models, as described in Section 4.2.3.

5.4.2 K-nearest Neighbors Baseline

In the absence of comparable studies for benchmarking my method, I developed a simple error-detection model based on the K-Nearest Neighbors (KNN) method, drawing inspiration specifically from the nearest-neighbor windows approach by Yu et al. (2014), as detailed in Section 2.1.2. However, this method was designed for real-valued signals, whereas my approach deals with sequences of categorical tokens, so I have modified it accordingly.

To adapt the nearest-neighbor window approach for categorical data, it is necessary to establish an *embedding* that converts categorical tokens into real-valued vectors within an n-dimensional space. This technique, common in the field of NLP, enables the definition of a distance metric between tokens. For effective KNN-based error detection, it is necessary that tokens with similar musical meanings are mapped to numerically proximate vectors, while highly dissimilar tokens are represented by vectors that are distant from one another. For instance, all quarter notes should be relatively close in the embedding space, with quarter notes positioned higher on the staff being slightly more distant from those lower down than from each other.

To achieve this, I employ the word2vec technique from the field of NLP (Mikolov et al. 2013). This method involves training a straightforward one-layer FNN on a task that requires filling

in missing tokens. This training process effectively teaches the network about the similarities between tokens based on how often they are substituted for each other in a dataset. For generating these token embeddings, I use the extensive dataset of correctly notated string quartets. The result of this training is a representation of each type of agnostic token as a 5-tuple of real values.

Having established the embeddings for each token, I next implement the nearest-neighbor windows method. In this approach, the context of each token is represented by concatenating the embeddings of the tokens preceding and following it. For instance, if the embedding dimension is set to 5, then the nearest-neighbor window of size 1 around any token will have a dimensionality of 10. Consider a sequence of embedded agnostic tokens T, representing an entire piece of music with length n. Let t_i be the embedding of the token at position i, and w_i be the nearest-neighbor window of size 1 centered on position i. The predicted error score $E(i)$ of the token at position i is then defined as

$$E(i) = ||t_i - t_j||, \text{ where } j = \arg\min_{x \leq n} ||w_i - w_x||. \tag{5.2}$$

Intuitively, the equation's right-hand side under the arg min operation asks the question: "Which other token in this sequence possesses a context most similar to the context of token i?" The equation's left-hand side compares distances between the tokens themselves, assigning a high value (indicative of a predicted error) when the tokens are dissimilar. This formulation initially considers only a single nearest neighbor. A more robust method involving any number of k nearest neighbors is shown in Equation 5.3.

$$E_k(i) = \frac{1}{k} \sum_{m=0}^{k} ||t_i - t_{j_m}||, \text{ where } j_0 ... j_k = \arg\min_{x_1, ... x_k} \sum_{m=0}^{k} ||w_i - w_{x_m}|| \tag{5.3}$$

This modification instead identifies the k closest nearest-neighbor windows to the window w_i, and takes the difference on the left to be the average distance from t_i to all the corresponding indices. This model is predicated on the assumption that tokens with similar contexts should themselves be similar. Under this framework, errors are defined as tokens that deviate from their expected contexts. For instance, if, in the entirety of some musical piece, all eighth notes are flanked on both sides by two eighth rests, none will be labeled as errors, as they all share a consistent context. However, should one of these eighth notes be adjacent to a different token, it would be identified as an error, given its deviation from the standard eighth note context.

5.5 Data Pipeline, Training, Testing, and Inference

Finally, in this section I describe how all of the mechanisms described in this chapter come together into a data preprocessing stage, a training loop, a testing procedure, and an inference procedure for the experiments in Chapter 6.

5.5.1 Data Preprocessing

The preprocessing phase entails transforming the symbolic music files with semantic encoding from both the OMR errors dataset (Section 5.3.1) and the synthetic dataset (Section 5.3.2). For both datasets, the entire set of MusicXML and humdrum **kern files undergoes the conversion process from semantic to agnostic, as described in Section 5.2.3.

For the synthetic dataset, the resulting agnostic sequences are saved in an intermediate data file. Since these sequences will have synthetic errors introduced during training, no additional preprocessing is needed. For the dataset of natural errors, I first pair up the agnostic representation of each piece in both its human-corrected and OMR output forms, and perform an ANW alignment between each pair. Using the results of this alignment, errors in the OMR-processed pieces are identified using the procedure described in Section 5.2.4. This procedure marks each token in the OMR-processed scores with a binary label of either 0 or 1, establishing the ground truth of error locations. A select few pieces from the dataset containing OMR errors are chosen for error analysis. This aids in training the mechanism responsible for introducing errors into the synthetic dataset, as outlined in Section 5.3.3.2. The rest of the pieces, now labeled with errors, are divided into training, validation, and testing subsets. These partitioned sets are then stored in a data file, ensuring they can be effortlessly incorporated into the model during the training phase.

5.5.2 Training, Validation, and Testing

The training process is summarized by the diagram in Figure 5.6. While training with synthetic data, errors are integrated into the dataset on a per-batch basis. This approach allows the integrated errors to vary with each batch, thus training the model with an effectively limitless volume of data. Each batch is subject to error incorporation using one of the three data augmentation strategies, potentially employing statistics related to OMR error features. The sequence containing errors is then matched against the original, correct sequence using an ANW alignment. This

produces a sequence of binary ground-truth labels that mark error positions within the sequence on a per-token level. The result is paired sets of input sequences and target sequences, which serve as training data for the model. When using the natural OMR errors for training, this step is not needed because the sequences were already aligned during the preprocessing phase.

During the model's training phase, the agnostic tokenization of each musical piece is zero-padded into a length that is a multiple of the input sequence length of the machine-learning model, so it can cleanly be divided into sequences of equal length. There are multiple possible methods of combining the synthetic and natural datasets for training; these are described in Section 6.1.1.1, which discusses the designs of specific experiments. In the majority of the experiments conducted, initial training is carried out on augmented synthetic data for several epochs and later undergoes fine-tuning on the smaller dataset containing natural OMR errors until the training process converges.

Regardless of whether the errors are natural or synthetic, the sequence containing errors is passed through the machine-learning model, resulting in a real-valued error prediction for each agnostic token. These predicted errors are then compared with the errors found by the ANW alignment, using the binary cross-entropy loss (Section 4.2.1), and the resulting loss score is used to update the weights of the model.

As shown in Figure 5.7, the validation step is identical to the training step but using a separate, smaller database of string quartets set aside for this purpose, and there is no need to update the weights of the model. Instead of the binary cross-entropy loss, I take the normalized recall score (Section 4.1.3) as a validation metric, so that I can optimize for this score instead. Training ends using an early stopping scheme, that detects when a certain number of epochs have passed without any improvement in the validation score.

Testing on OMR data is identical to the validation procedures in Figure 5.7. For every test case, I fetch a piece in agnostic format containing natural OMR errors along with its corresponding per-token error labels. From there, the process is the same as with the validation loop.

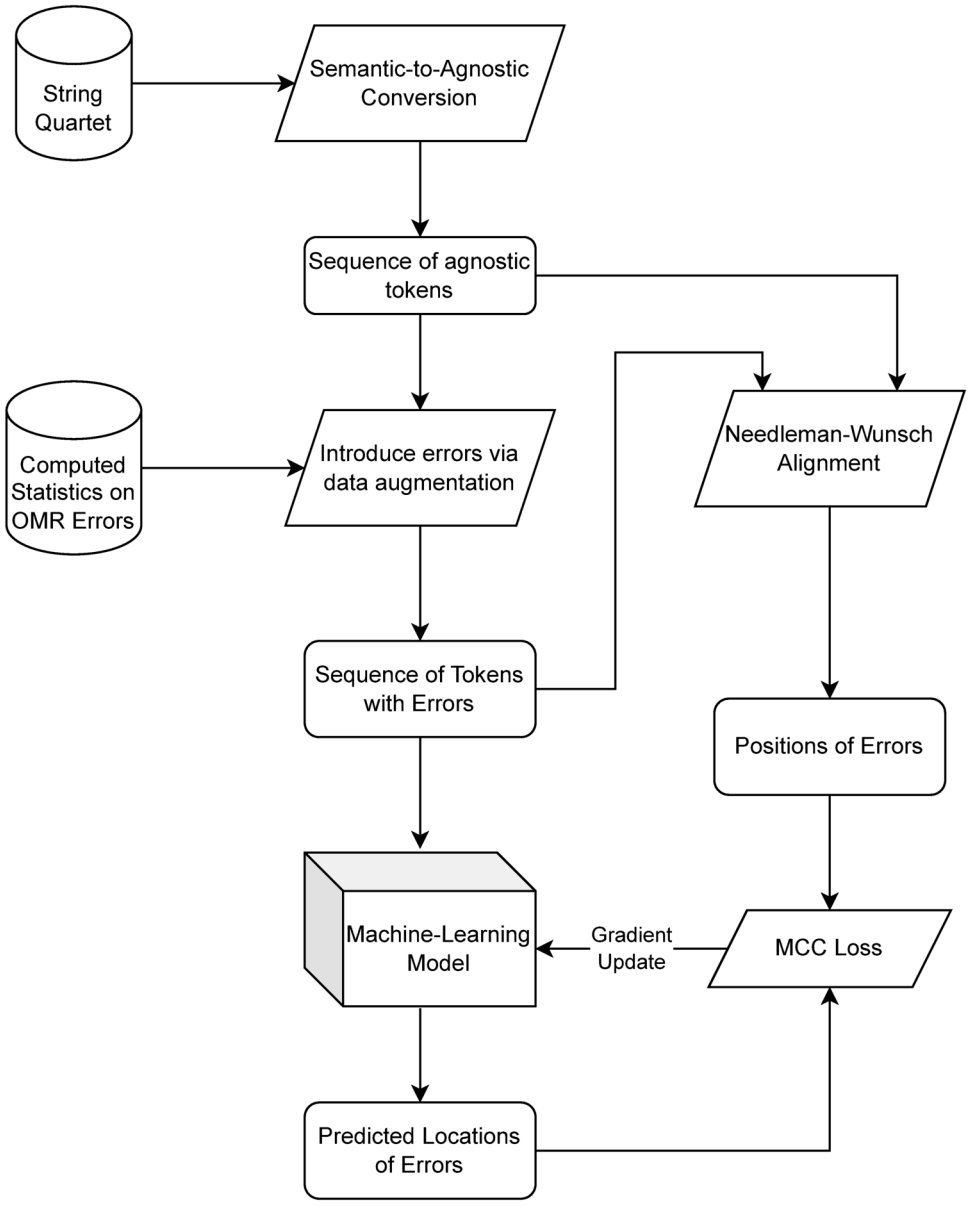


Figure 5.6: A figure displaying the process of training the error detector on synthetic errors. This entire process is executed once per training batch, so that every new batch sees newly generated artificial errors.

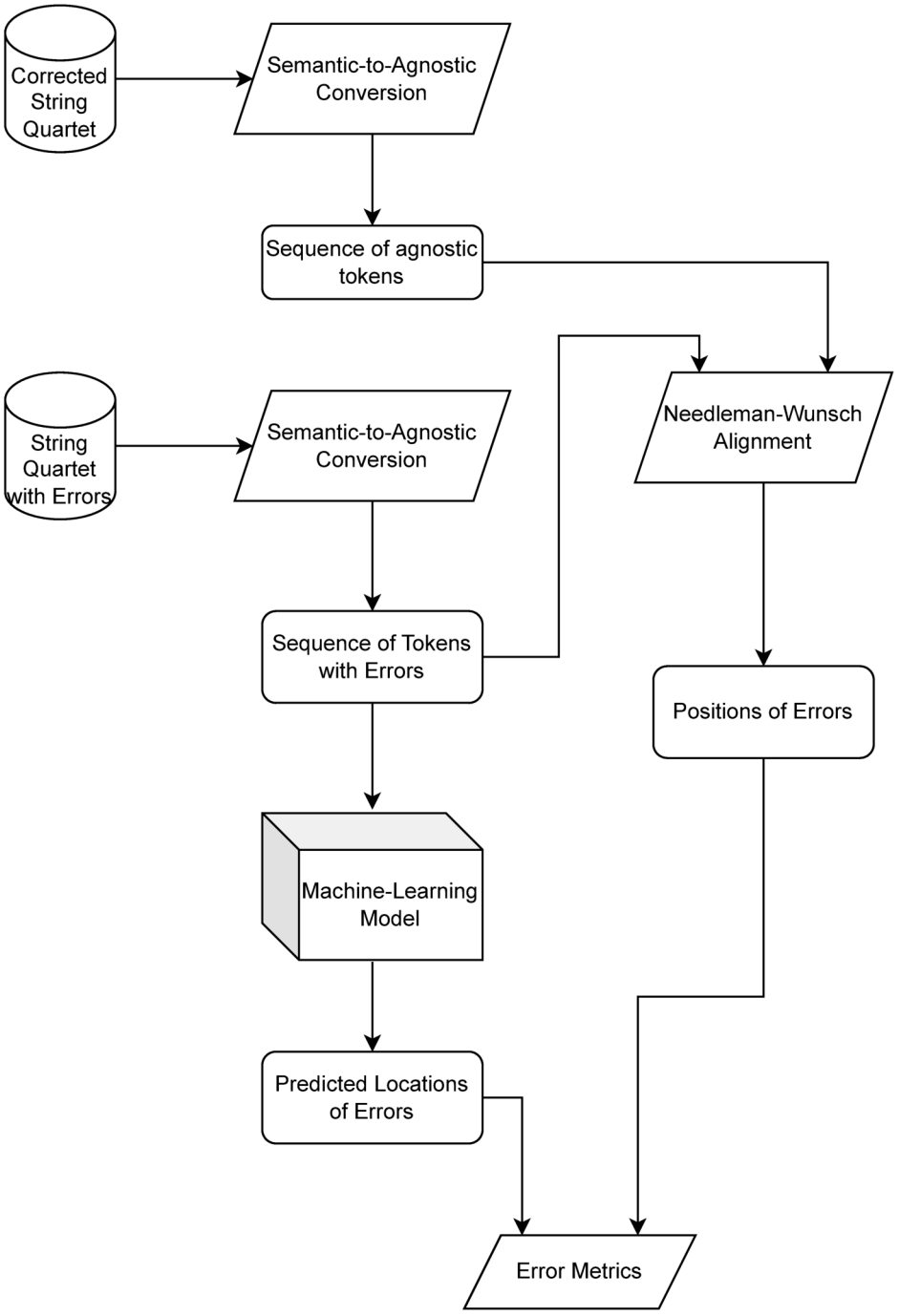


Figure 5.7: A figure displaying the process of validating the error detector during training, or testing its performance on real OMR data.

5.5.3 Inference and Visualization Procedure

The model's binarized predictions are not a user-friendly way to interact with the results. To be functionally beneficial, the error detector requires a method of visualizing predictions as annotations on a score, and this visualization process is more complex than the validation and testing procedures. Here I describe a process for running the model end-to-end, starting from a semantically encoded symbolic music file containing OMR errors and ending with an annotated version of that file. This process is summarized by the diagram in Figure 5.8.

The initial step in the inference workflow involves converting a user-provided symbolic music file into an agnostic format. Simultaneously, an agnostic-to-semantic alignment is maintained. This alignment registers the association between elements of the semantic input and the corresponding tokens (Refer to Section 5.2.3). The sequence of agnostic tokens is then padded to a multiple of the model's input sequence length and then divided into uniformly sized sequence batches. These batches are then fed into the machine-learning model to obtain raw-valued predictions. The model's continuous predictions are thresholded based on a user-specified sensitivity parameter, leading to binary predictions for each input token.

The agnostic sequence of tokens is then cross-referenced with the agnostic-to-semantic alignment and the original semantic music file to determine which elements of the semantic file should be marked as having errors. The output is a MusicXML file where every glyph marked as an error by the method is tagged with a specific color using the format's `color` attribute.

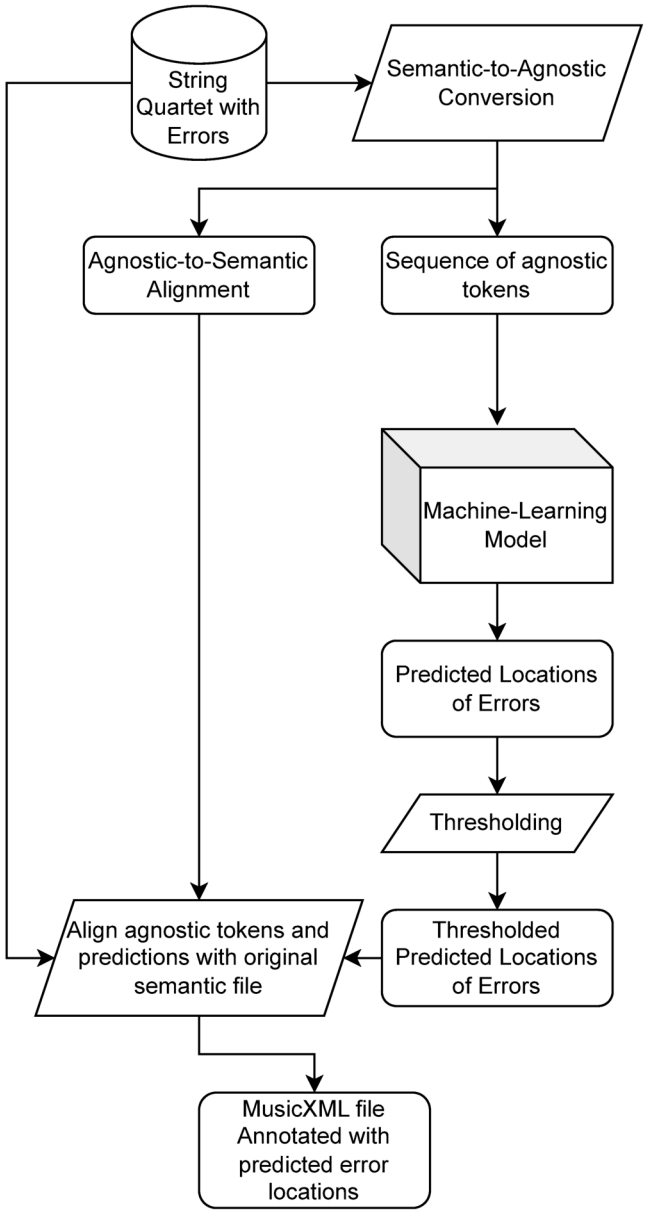


Figure 5.8: A figure displaying the process of inference on unseen semantically encoded musical scores containing OMR output.

Chapter 6

Results and Discussion

In this chapter, I present the outcomes of experiments conducted using the error-detection framework and datasets established in Chapter 5. First, in Section 6.1 I describe the experimental methodology, organizing the trials into three main categories: alterations to the machine-learning architecture, changes to data augmentation techniques and their integration into the training process, and variations in the input sequence length. Section 6.1.2 details the results for each set of trials along with pertinent training metrics. Section 6.2 provides engraved long-form examples of musical scores annotated by the model's output. Section 6.3 concludes the chapter with an evaluation of the error detection model's strengths and weaknesses, along with its potential to expedite human correction tasks.

6.1 Evaluation

To evaluate my error-detection system I will train the model under a variety of different circumstances and then evaluate the performance of the model on the same task. I refer to every full run of the training and evaluation process as a *trial*. Each trial has a shorthand title for ease of reference, given in Table 6.1. The same table shows, for each trial, exactly what parameters of it have been changed from the "Base" model, which acts as a default set of parameters (see Table 5.5 for an explanation of the tests that were run to arrive at this default set). Some hyperparameters are the same for all trials; these are shown in Table 6.2.

Table 6.1: A table defining all of the trials, separating them into four categories. The "Base" model is taken as a default set of hyperparameters, and other trials use variations on this set.

Trial	Architecture	Data Augmentation	Dataset	Trainable Parameters	Positional Encoding	Batch Size	Input Length	Max. time (h)
Base Model	LSTUT	Full	Fine-tuned	8085761	N	192	512	8
Architectures:								
LSTM	LSTM	Full	Fine-tuned	4408321	N	192	512	8
TF	Vanilla Transformer	Full	Fine-tuned	15964161	Y	192	512	8
UT	Universal Transformer	Full	Fine-tuned	4931841	Y	192	512	8
KNN	K-Nearest Neighbors	N/A	No Synthetic Errors	0	N/A	–	–	–
Augmentation:								
50% Errors	LSTUT	50% Fewer Errors	Fine-tuned	8085761	N	192	512	8
No Heuristics	LSTUT	No Clustering	Fine-tuned	8085761	N	192	512	8
Simple Data Aug	LSTUT	Random	Fine-tuned	8085761	N	192	512	8
No Fine-Tuning:								
Combined	LSTUT	Full	Combined	8085761	N	192	512	8
OMR Data Only	LSTUT	N/A	No Synthetic Errors	8085761	N	192	512	8
Synthetic Only	LSTUT	Full	No Natural Errors	8085761	N	192	512	8
Sequence Length:								
Length: 64	LSTUT	Full	Fine-tuned	8085761	N	1536	64	8
Length: 128	LSTUT	Full	Fine-tuned	8085761	N	768	128	8
Length: 256	LSTUT	Full	Fine-tuned	8085761	N	384	256	8
Length: 1,024	LSTUT	Full	Fine-tuned	8085761	N	96	1024	8
Length: 2,048	LSTUT	Full	Fine-tuned	8085761	N	32	2048	12
Length: 4,096	LSTUT	Full	Fine-tuned	8085761	N	16	4096	12

6.1.1 Experiment Design

Training the models I have defined takes many hours and requires a significant amount of computational resources. The architecture of the Long Short-Term Universal Transformer (LSTUT) has many hyperparameters, and the data augmentation procedures I define have many more; it would be impractical to try every possible combination of parameters to find the ideal model. In addition, different architectures and hyperparameters induce training processes that take significantly more memory or more time per epoch, so fully automating parameter sweeps would be difficult for most hyperparameters.

I use two strategies to narrow down the possible values for sets of hyperparameters to test. The first is to base the evaluation of these models around maximizing utilization of the GPU resources available to me. The largest Graphics Processing Unit (GPU) nodes I have access to that I can perform large-scale tests on are nodes hosted on the Digital Alliance of Canada[1] that contain four NVIDIA Tesla P100 Pascal[2] GPUs, each containing 12 gigabytes of GPU memory, for a total of 48 GB across the whole node. Each set of parameters that I test will be chosen so as to use just under 48 GB of GPU memory in training. The second strategy is to use a "Base" set of fairly optimal parameters and have all tests be minimal variations on this "Base" model, to gauge the relative importance of each hyperparameter separately. This approach assumes that each hyperparameter influences the model's performance in a linear and independent manner, which is a common assumption when performing hyperparameter searches (See Section 4.2.3).

6.1.1.1 Experimental Setup: Training Schedule

For all trials, models are tr ained with the Adam optimizer (Kingma and Ba 2017), using its implementation in the PyTorch machine-learning framework, and its default settings. The loss function used is binary cross entropy (See Section 4.2.1), with positive examples given extra weight to compensate for the class imbalance between erroneous and correct tokens in the data.

Training employs a cyclic learning rate scheduler (See Section 4.2.3). The learning rate cycles between 0 and a maximum value of 0.01, and the cycle's maximum value is halved after each subsequent cycle. The time to complete each cycle is counted in batches rather than epochs, which is why the cycle rate does not appear uniform when graphed against epoch number. See Figure 6.1 for an illustration of how the learning rate changes over time. Batch size is adjusted

1. alliancecan.ca/en/services/advanced-research-computing
2. nvidia.com/en-us/data-center/tesla-p100/

Table 6.2: Hyperparameter values common to all trials defined in Table 6.1.

Hyperparameter	**Default Value**
Model dimension	512
Model dropout	0.2
LSTM hidden dimension	512
Transformer feed-forward layer width	2048
Attention heads per Transformer layer	12
LSTUT depth	6
Max # epochs	100
Max. # epochs training on augmented data	30
# Epochs for early stoppping	30
Maximum learning rate	0.01
Length of learning rate cycle (in batches)	150

for each trial to yield a roughly equal number of batches per epoch; however, trials with shorter input sequence lengths still result in fewer batches per epoch. This discrepancy arises because each string quartet's agnostic representation is padded to a multiple of the input length, ensuring that the model does not receive inputs from multiple pieces simultaneously. This padding leads to slight variations in the total number of tokens in the training data across trials, affecting the cycle rates shown in Figure 6.1. The reduction in learning rate variation after epoch 30 corresponds to a switch to fine-tuning on a smaller dataset, causing a lower number of batches per epoch.

Training uses an early-stopping scheme that stops training when the validation score ceases to increase for a set number of epochs, and the model saved after the training process completes is the one that achieved the highest validation score on real Optical Music Recognition (OMR) data (for runs that use the real OMR data). This process is modified for some of the runs that alter the mechanics of the data augmentation process (see Section 6.1.1.5).

6.1.1.2 Experimental Setup: Architecture

Four experimental runs detailed in Table 6.1 investigate architectures distinct from the LSTUT. Due to their unique hyperparameters and memory consumption patterns, these require special consideration in the following experiments.

The "UT" trial trains with a Universal Transformer model, which is identical to the Long Short-Term Universal Transformer (LSTUT) but without the Long Short-Term Memory (LSTM) layers at its beginning and end. The "TF" trial uses a vanilla Transformer decoder model, as outlined in Section 4.3.3. A significant characteristic of the model used in the "TF" trial is

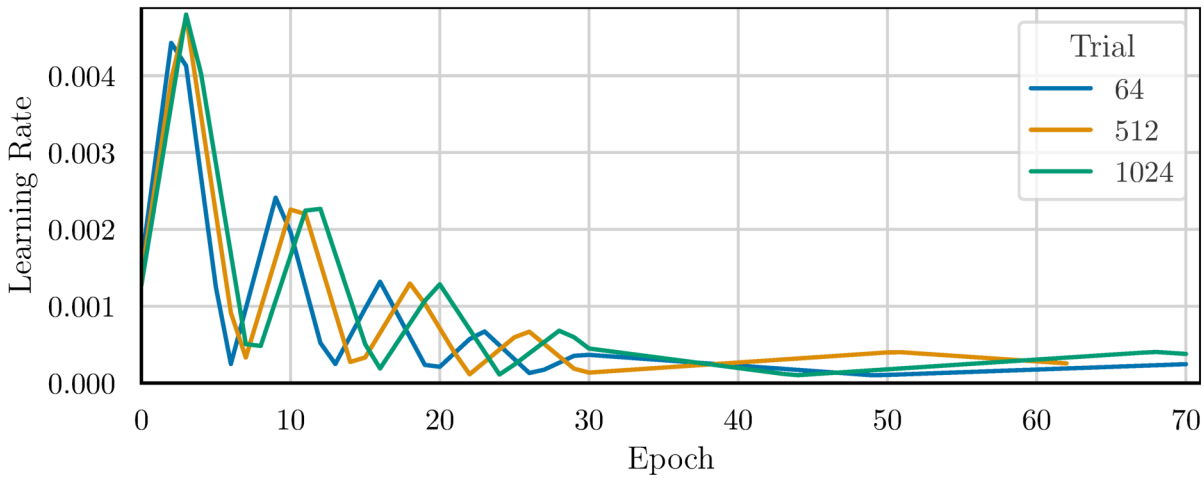


Figure 6.1: The learning rate per epoch for a few of the trials on varying input sequence lengths. Only three trials are shown for visual clarity, as the graphs of the others overlap these three heavily.

its considerably larger number of trainable parameters compared to other architectures. This is because the Universal Transformer (UT), central to the LSTUT, utilizes four transformer layers that share the same set of trainable parameters. When the weights of these layers are uncoupled, the UT's trainable parameters effectively increase by a factor equal to the number of layers. Despite their differences in parameters, the "UT" and "TF" runs have equal memory requirements during training. This is because the memory requirement during backpropagation is contingent on the number of operations executed on the network input, not the number of distinct parameters involved. Adhering to my principle of maximizing compute utilization for each trial, the "TF" run retains its elevated parameter count, ensuring it consumes GPU memory comparable to other Transformer-based designs. Like the "Base" run, both "UT" and "TF" integrate low-rank attention to optimize memory. They both employ a positional encoding method, compensating for the lack of LSTMs present in the LSTUT.

The "LSTM" trial trains a two-layer bidirectional LSTM as model, effectively using the "Base" model's LSTUT without the Universal Transformer. LSTMs are not nearly as memory-intensive as Transformers are, so increasing the number of layers of the LSTM only slightly increases the GPU memory usage; it does, however, linearly increase the amount of time that it takes to a perform an epoch. Utilizing all 48 GB of GPU memory available with just a deep LSTM would result in a model with an high number of parameters that takes impractically long to train, so I leave this architecture at two layers.

The "KNN" trial implements a basic non-parametric model based on K-Nearest Neighbors (KNN), leveraging the nearest-neighbor windows technique presented by Yu et al. (2014) and

described in Section 4.1.1. Each token's nearest-neighbor window consists of the tokens before and after it, effectively representing its context. This model is predicated on the assumption that tokens with similar contexts should themselves be similar. Under this framework, errors are defined as tokens that deviate from their expected contexts. For instance, if, in the entirety of some musical piece, all eighth notes are flanked on both sides by two eighth rests, none will be labeled as errors, as they all share a consistent context. However, should one of these eighth notes be adjacent to a different token, it would be identified as an error, given its deviation from the standard eighth note context in that piece. The model yields a singular real value for each token, indicating the distance between the token and its most contextually similar points. This distance value is then subjected to thresholding, as with other methods. A breakdown of this method's operations and the token similarity computations can be found in Section 5.4.2. Given its lack of trainable parameters, this method is used here as a "sanity check" baseline.

6.1.1.3 Experimental Setup: Sequence Lengths

Several trials compare the effect of differing input sequence lengths. As the Transformer architecture does not require specifying the input sequence length, the models of these runs are all identical. Since I am using a variation of the Transformer architecture that implements low-rank attention (Section 4.3.4.3), an increase in the input sequence length results in an increase of the same magnitude in memory needed during training. For this reason, in runs that employ input sequence lengths shorter than the Base model's length of 512, I increase the batch size by the same factor to maintain the same overall memory usage. This also ensures that across all these trials, every epoch goes through the training data in approximately the same number of batches, which is important for consistent behavior from the cyclic learning rate scheduler.

For the trials with lengths 2048 and 4096, I decrease the batch size by a slightly larger factor than should be necessary. When training a model with a batch size of 48 and a sequence length of 2048, training would occasionally fail from an out-of-memory error, sometimes after several epochs had already elapsed. The implementation of low-rank attention and the machine-learning framework that I am using appear to incur some additional memory overhead when processing long sequence lengths, and so the memory usage is slightly higher than the value predicted by theory. Reducing their batch sizes below the pattern established by the other trials ensures that they do not fail. To compensate for the increased number of batches necessary to cover the entire dataset, I also increase the maximum allowable time that they are allowed to train, so

that they can be allowed as many training epochs as the other trials.

6.1.1.4 Experimental Setup: Datasets

In Section 5.3, I detailed the composition of the dataset used for evaluating the musical error detector. This dataset consists of two distinct parts: a smaller dataset containing natural OMR errors and a larger dataset utilized for generating synthetic training data.

The smaller dataset contains string quartets with natural OMR errors, which are generated by processing the score images through PhotoScore's OMR system. Each of these quartets are paired with a human-corrected version from either the Mendelssohn String Quartet Dataset (MSQD) or the OpenScore String Quartets (OSSQ) corpus. This dataset provides examples of natural OMR errors and their corrected versions for model training and validation. The larger dataset comprises a collection of human-corrected string quartets, which do not have corresponding versions with OMR errors. This dataset is used for data augmentation, as outlined in Section 5.3.3, where synthetic OMR-like errors are added to the correct scores. The augmentation process is designed to be efficient, allowing for the generation of new errors for each training epoch, thereby enhancing the robustness and generalizability of the trained model.

Both the small and large datasets are divided into training, validation, and test splits, with proportions of 75%, 10%, and 15% respectively. Musical material tends to repeat throughout individual compositions. To avoid overlap of musical material in the training and test sets, which could compromise the integrity of the evaluation, the splitting process ensures that each segment of the dataset corresponds to at least a single movement of a quartet, and individual movements are not split into smaller pieces.

In the "OMR data only" trial, the model is trained exclusively on the small dataset containing natural OMR errors, without any use of data augmentation. Conversely, the "Synthetic only" trial involves training solely on data augmented with synthetic errors, utilizing the larger dataset. The "Combined" trial represents a straightforward approach to combining these datasets, where both natural and synthetic datasets are merged into a larger composite dataset, and the model is trained on a combination of real OMR errors and synthetic errors simultaneously.

In contrast, the "Base" trial and all other trials employ a fine-tuning strategy. Initially, the model is trained on the synthetic data from the larger dataset until it reaches a certain performance threshold or a maximum number of epochs have elapsed (30, as specified in Table 6.2). Following this initial training phase, the model is further trained (fine-tuned) on the natural

OMR errors from the smaller dataset. This approach aims to leverage the generality learned from the synthetic data and then refine the model's understanding with real-world OMR error examples.

During the fine-tuning phase, the model's performance is validated on the natural OMR errors from the small dataset's validation partition. The large dataset's test partition, which contains unseen music pieces, is used to generate additional synthetic errors. These serve as a control during evaluation, ensuring that the model can reliably detect synthetic errors and thus demonstrating its basic competence in error detection.

6.1.1.5 Experimental Setup: Data Augmentation

For trials that use synthetic data, the augmentation processes outlined in Section 5.3.3 are employed. The "Base" trial uses the probability-based augmentation method defined in Section 5.3.3.3 that uses statistics about natural OMR errors and heuristics to generate synthetic errors with a similar distribution to real errors. I also include trials that vary parameters of the data augmentation (see Table 6.1). The "No Heuristics" trial disables the heuristics that force generated errors to clump together. The "Simple Data Aug." trial uses only simple data augmentation (See Section 5.3.3.1) without using statistics on natural OMR errors. In the test set, 21% of the tokens are marked as errors, so for this trial the per-token modification probability is set to 0.21. Lastly, the "50% errors" trains with the full data augmentation process, utilizing both statistics of real errors and heuristics, but reduces the number of generated errors by half by modifying the analyzed probabilities (see Table 5.4).

6.1.2 Results

Table 6.3 contains a summary of the results of all trials, organized by category. The primary metric for comparing trials is the Normalized Recall (R_{norm}), chosen for its utility in high-recall information retrieval and its absence of a threshold requirement. The R_{norm} score directly evaluates the model's output, which is a ranking of agnostic tokens based on their likelihood of being errors. An R_{norm} of 1 represents a perfect ranking where all erroneous tokens are ranked above all correct tokens, whereas a score of 0 represents the inverse. If an erroneous token happens to be ranked lower than some correct ones, the R_{norm} score decreases, and it decreases further the lower the erroneous token is ranked. In this way, the score evaluates not just how correct the model's predictions are, but how confident the model is in them. A highly confident

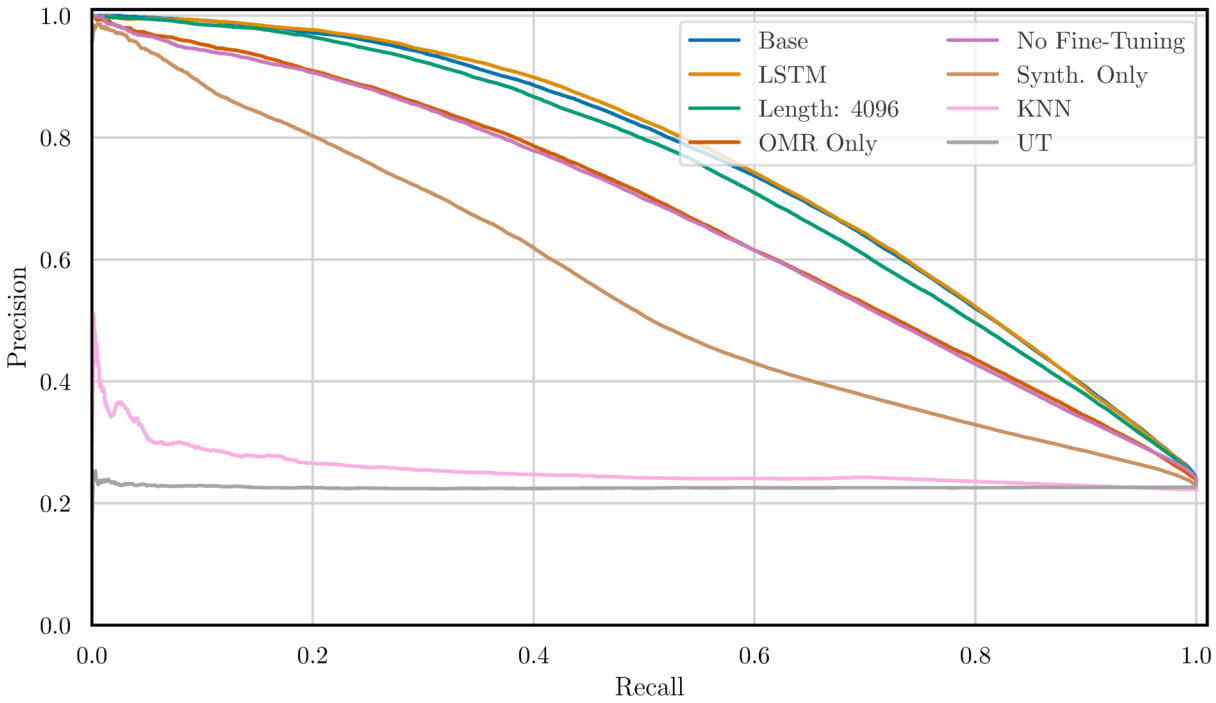


Figure 6.2: The Precision-Recall curves of selected trials. Each of these lines is parametrized by the threshold used to binarize the data into predictions; each point on a specific trial's curve represents a pair of possible Recall and Precision scores. This smaller subset of trials was chosen for readability; most of the curves lie very close to the curve for the "Base" trial.

yet incorrect prediction will have a greater negative impact on the R_{norm} score than an incorrect prediction with a borderline score.

I also include two additional metrics: the Average Precision (AP) and the Precision at 0.9 Recall. The average precision, reviewed in Section 4.1.3, is a somewhat less robust measure than R_{norm}, as it does not take the overall size of the dataset into account, and only the number of positive (erroneous) elements. However, it is simpler and more widely used than R_{norm} across information retrieval tasks, and it has the intuitive interpretation of being equal to the area under the precision-recall curve (see Figure 6.2 for graphs of the precision-recall curve for several trained models). To calculate the precision at 0.9 recall, I find the threshold that produces a recall score of 0.9 when applied to the validation set of natural OMR errors, and use that threshold to binarize the model's predictions on the test set. In the context of a human-interactive error detection setting, this metric could be interpreted as "the fraction of predicted errors that are actual errors when the interface is configured to miss at most one in ten errors."

Table 6.4 shows additional measurements made on each trial to illustrate the model's performance at high recall rates. The **Prop. Tokens Marked** column here corresponds to the proportion of the score marked as erroneous by the model. This statistic was referred to as e in

Table 6.3: A table containing summary metrics describing the performance of each trial run. Groups of rows correspond to the categories of trials defined in 6.1. Bolded entries in each column correspond to the trial with the best performance in that metric. The performance metrics used here are defined in Section 6.1.2.

Trial	**Synthetic Training (h)**	**Fine-Tuning Time (h)**	**Total Training Time (h)**	**AP**	**R_{norm}**	**Precision at 0.9 Recall**
Base Model	4:55	0:44	5:40	0.74	**0.87**	**0.39**
Architectures:						
LSTM	4:32	0:37	5:10	**0.75**	0.84	0.37
TF	2:28	0:22	2:51	0.21	0.46	0.23
UT	2:21	0:22	2:43	0.23	0.50	0.23
KNN	-	-	-	0.25	0.53	0.23
Data Aug.:						
50% Errors	4:51	0:48	5:39	0.73	0.87	0.38
No Heuristics	4:58	1:02	6:00	0.74	0.87	0.38
Simple Data Aug.	7:33	0:27	8:00	0.72	0.85	0.38
No Fine-Tuning:						
Combined	-	-	8:03	0.67	0.84	0.34
OMR Data Only	-	0:59	0:59	0.65	0.83	0.33
Synthetic Only	8:00	-	8:00	0.53	0.74	0.23
Sequence length:						
Length: 64	5:52	0:55	6:48	0.73	0.86	0.37
Length: 128	4:37	0:41	5:18	0.74	**0.87**	0.38
Length: 256	4:22	0:42	5:04	0.74	**0.87**	0.38
Length: 1,024	6:31	0:55	7:27	0.72	0.86	0.37
Length: 2,048	8:46	0:42	9:29	0.72	0.85	0.36
Length: 4,096	8:45	0:35	9:20	0.67	0.84	0.34

Equation 3.6, which calculates an estimate for time saved by the error detector. The lower this value is for any given recall value, the more the detector is able to narrow down its predictions to a smaller area of the score.

I also supply figures showing training loss, validation loss, and validation R_{norm} for the trials in each category over the course of the training process. Figure 6.3 shows pre-training and fine-tuning metrics for the "Architectures" trials, Figure 6.4 shows the same metrics for the "Data Aug." trials, and Figure 6.5 shows metrics for the "Sequence Length" trials. For these sets of trials, the first three figures show metrics during pre-training, while the last three show metrics when fine-tuning on natural errors. Figure 6.6 shows only three graphs of training metrics for the "No Fine-Tuning" category of trials, since trials in this category are trained in only one step. As explained above in Section 6.1.1.1, during both pre-training and fine-tuning, training stops when either a maximum number of epochs is reached or when the model ceases to improve on its validation metric. As a result, some of the lines in these figures cut out in the middle, if they have reached the convergence criteria early.

For the remainder of this section, I discuss the results on each category of trials, as defined in Table 6.1.

6.1.2.1 Discussion of Results using Different Architectures

As shown in Table 6.3, the vanilla Transformer and Universal Transformer significantly underperformed compared to the LSTUT architecture used in the "Base" trial. Their performance is only slightly better than an error detection strategy that labels all input as erroneous. In most trials, about 21% of the tokens in the test set are positive, meaning an error predictor marking all tokens as errors would achieve a minimum precision score of 0.21.[3] As shown in Table 6.4, the Transformer and Universal Transformer correctly identify non-erroneous tokens only 1% and 11% of the time, respectively, at 0.9 recall.

The LSTM model performs better than the others, approaching the performance of the LSTUT model used in the "Base" trial, and slightly outperforming them on the metric of average precision. Figure 6.2 shows that this is due to its slightly better performance at very low recall values; it performs the same as or slightly worse than the "Base" model at high recall.

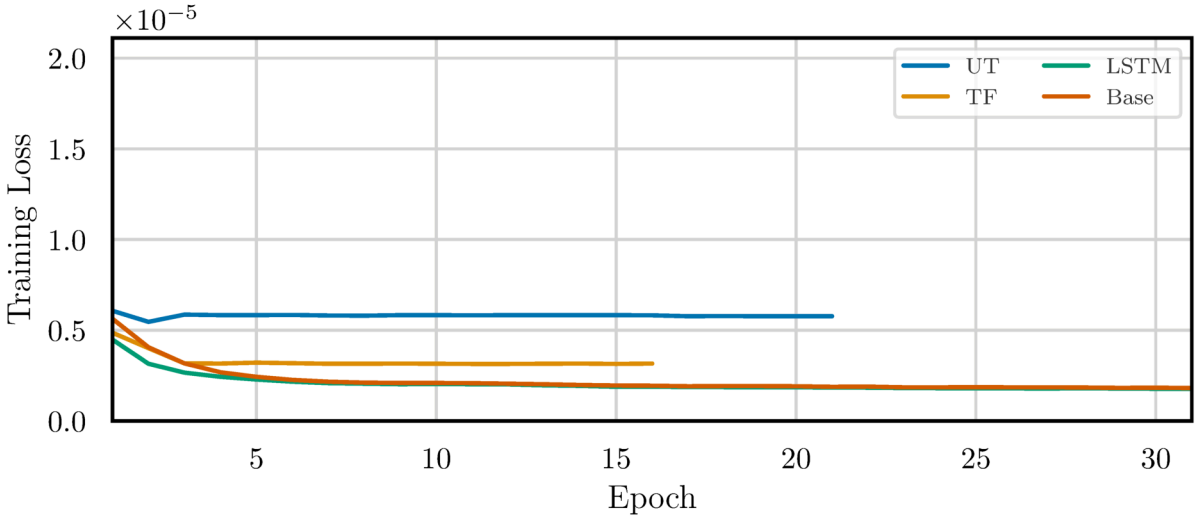
3. The precise number varies slightly between trials because performing the Needleman-Wunsch alignment on string quartets that have been partitioned into segments of differing lengths creates slightly different alignments, some with more or less errors. In addition, different sequence lengths require different amounts of padding for each string quartet, as discussed in Section 6.1.1.1.

Table 6.4: A table containing measurements of each trial at particular high recall rates. The "Prop. Tokens Marked" notes how many of the predictions of the model are positive when achieving the particular recall rate listed above. The "True Neg. Rate" column notes what percentage of non-erroneous tokens are correctly identified as not errors. Bolded entries in each column correspond to the trial with the best performance in that metric.

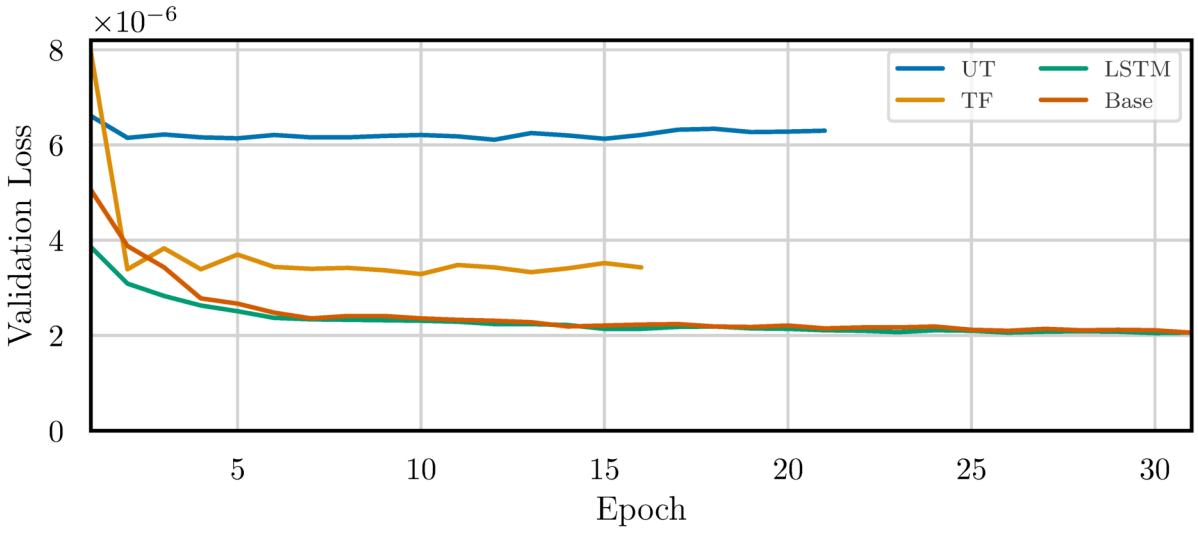
	At 0.9 Recall			**At 0.99 Recall**		
Trial	**Precision**	**Prop. Tokens Marked**	**True Neg. Rate**	**Precision**	**Prop. Tokens Marked**	**True Neg. Rate**
Base Model	**0.39**	**0.53**	**0.58**	**0.26**	0.87	**0.17**
Architectures:						
LSTM	0.37	0.54	0.55	**0.26**	0.87	0.16
TF	0.23	0.98	0.01	0.23	0.98	0.01
UT	0.23	0.89	0.11	0.23	0.98	0.02
KNN	0.23	1.00	0.00	0.23	1.00	0.00
Data Aug.:						
50% Errors	0.38	0.54	0.57	0.26	0.87	0.17
No Heuristics	0.38	0.54	0.57	0.26	0.88	0.16
Simple Data Aug.	0.38	**0.53**	0.58	0.26	0.87	0.16
No Fine-Tuning:						
Combined	0.34	0.60	0.48	0.26	0.88	0.16
OMR Data Only	0.33	0.61	0.47	0.25	0.89	0.14
Synthetic Only	0.23	1.00	0.00	0.23	1.00	0.00
Sequence Length:						
Length: 64	0.37	0.55	0.55	**0.26**	**0.86**	**0.17**
Length: 128	0.38	**0.53**	**0.58**	**0.26**	**0.86**	**0.17**
Length: 256	0.38	**0.53**	0.57	**0.26**	**0.86**	**0.17**
Length: 1,024	0.37	0.55	0.56	0.25	0.87	**0.17**
Length: 2,048	0.36	0.57	0.53	0.23	1.00	0.00
Length: 4,096	0.34	0.60	0.49	0.25	0.88	0.16

Table 6.5: A table containing summary metrics describing each trial run on other test sets. Each of these evaluations are run after the relevant model has been trained to completion on the training set of natural OMR errors (for those trials that use natural OMR errors). The evaluations on synthetic data are performed on the data generated using the trial's particular data augmentation parameters; so, the "Simple Data Aug." trial is evaluated on simple, randomly augmented data, and so on. Bolded entries in each column correspond to the trial with the best performance in that metric. The performance metrics used here are defined in Section 6.1.2.

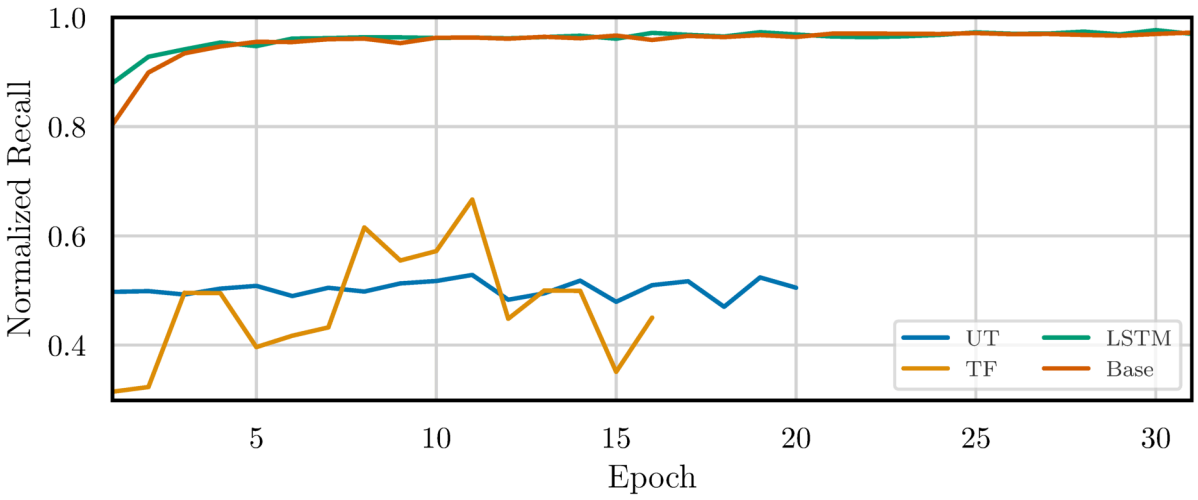
	On Partially Corrected OMR			**On Synthetic Data**		
Trial	**AP**	**R_{norm}**	**Precision at 0.9 Recall**	**AP**	**R_{norm}**	**Precision at 0.9 Recall**
Base Model	0.30	**0.79**	**0.09**	0.82	0.93	0.54
Architectures:						
LSTM	**0.36**	0.78	0.08	0.82	0.93	0.55
TF	0.06	0.51	0.06	0.22	0.49	0.22
UT	0.06	0.51	0.06	0.88	0.41	0.86
KNN	0.09	0.51	0.08	-	-	-
Data Aug.:						
50% Errors	0.26	0.78	0.08	0.62	0.94	0.26
Simple Data Aug.	0.24	0.78	0.08	0.88	0.94	0.54
No Heuristics	0.26	0.76	0.08	0.83	0.90	0.52
No Fine-Tuning:						
Combined	0.16	0.71	0.08	0.90	0.96	0.72
OMR data only	0.20	0.74	0.07	0.48	0.79	0.33
Synthetic only	0.07	0.49	0.06	**0.93**	**0.98**	**0.78**
Sequence Length:						
Length: 64	0.28	0.78	**0.09**	0.83	0.93	0.58
Length: 128	0.26	**0.79**	**0.09**	0.85	0.94	0.61
Length: 256	0.26	**0.79**	**0.09**	0.81	0.93	0.53
Length: 1,024	0.21	0.75	0.08	0.81	0.92	0.52
Length: 2,048	0.24	0.76	0.09	0.80	0.92	0.52
Length: 4,096	0.20	0.78	0.07	0.81	0.94	0.61



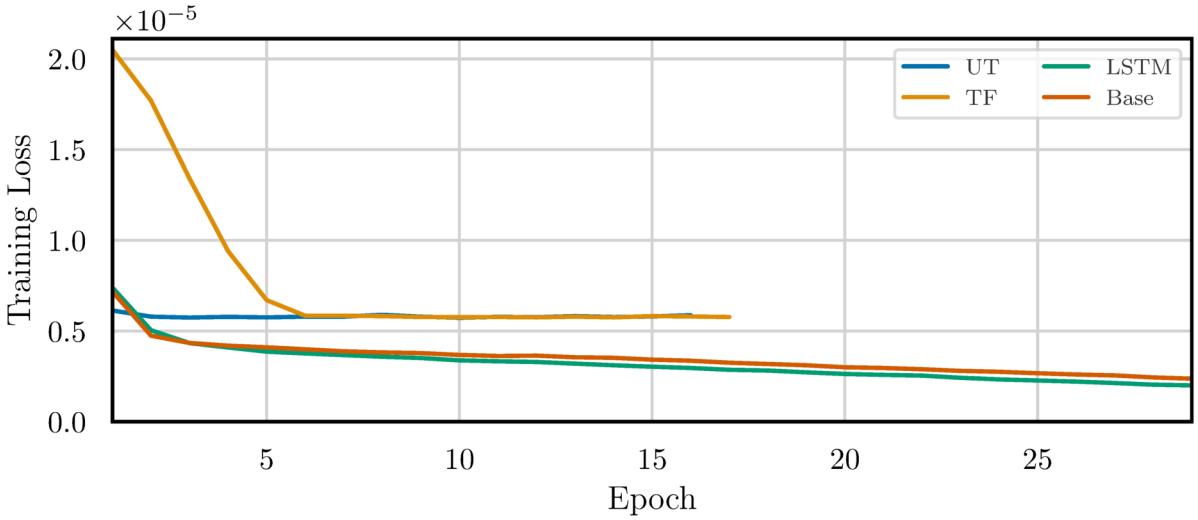
(a) The training loss of each trial over each epoch, during pre-training on synthetic errors.



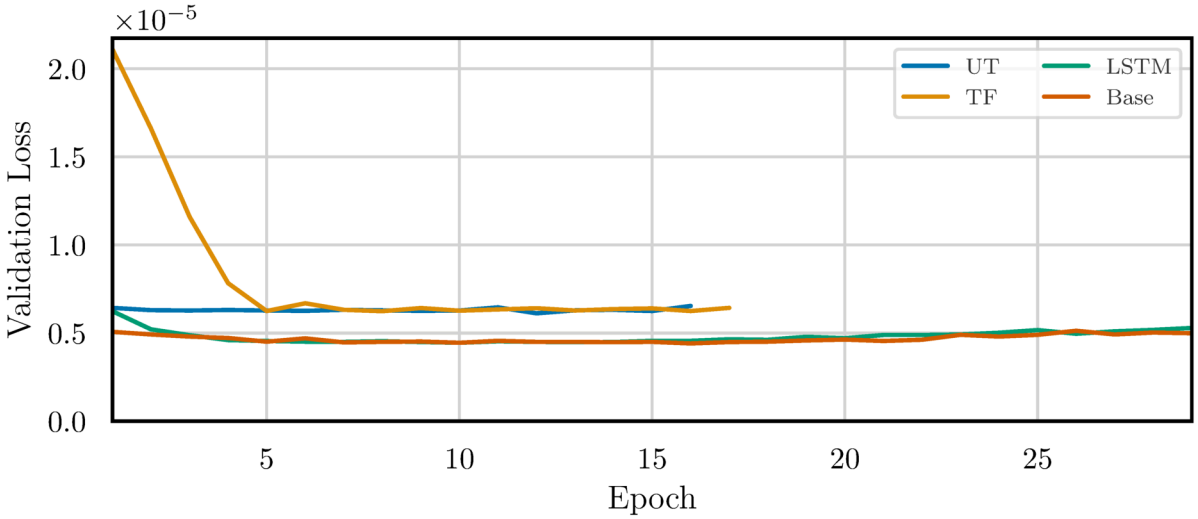
(b) The loss on the validation set on each trial over each epoch, during pre-training on synthetic errors.



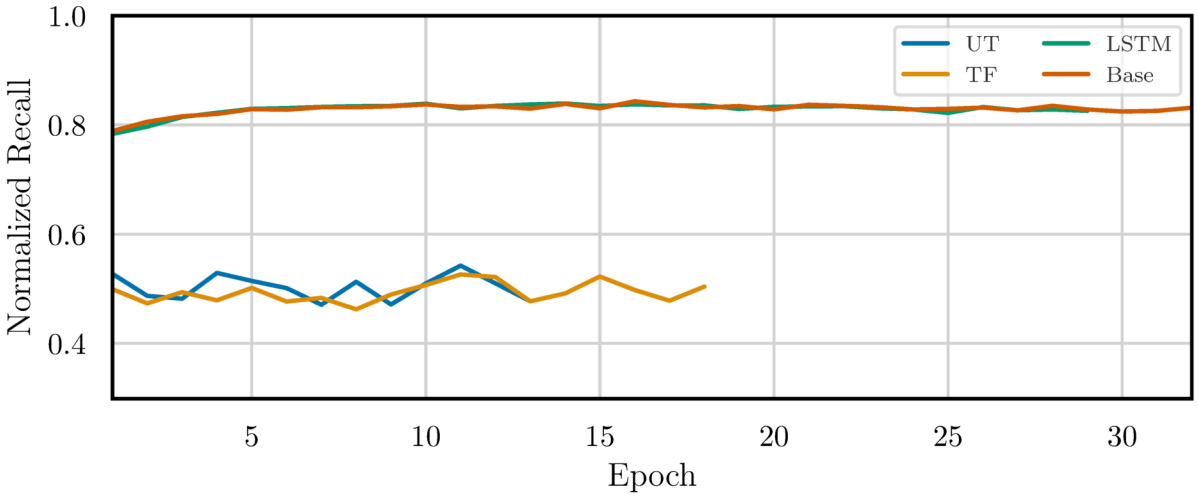
(c) The normalized recall on the validation set of each trial over each epoc, during pre-training on synthetic errors.



(d) The training loss of each trial over each epoch, during fine-tuning on real OMR errors.

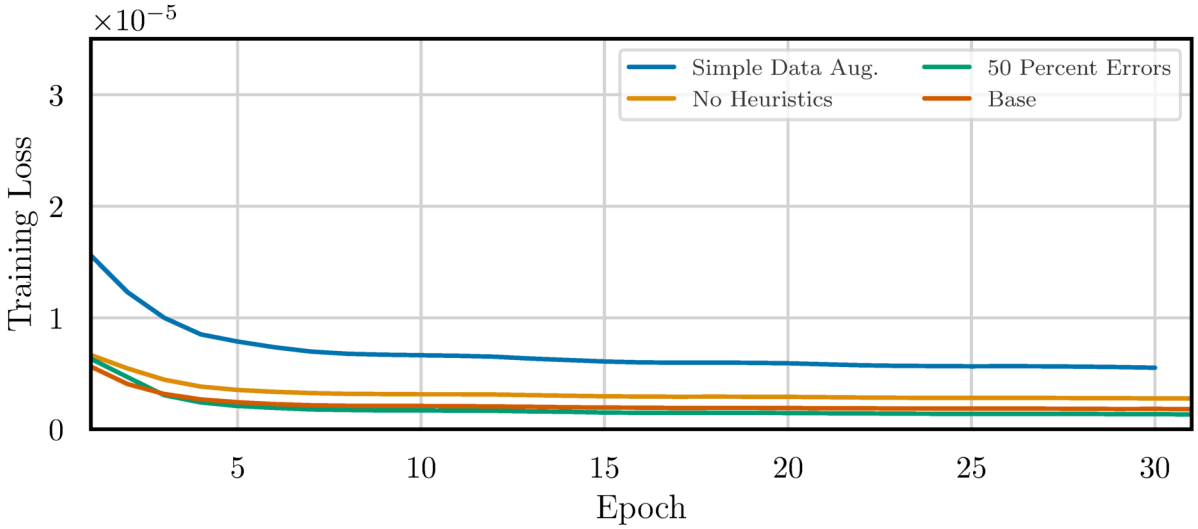


(e) The loss on the validation set on each trial over each epoch, during fine-tuning on real OMR errors.

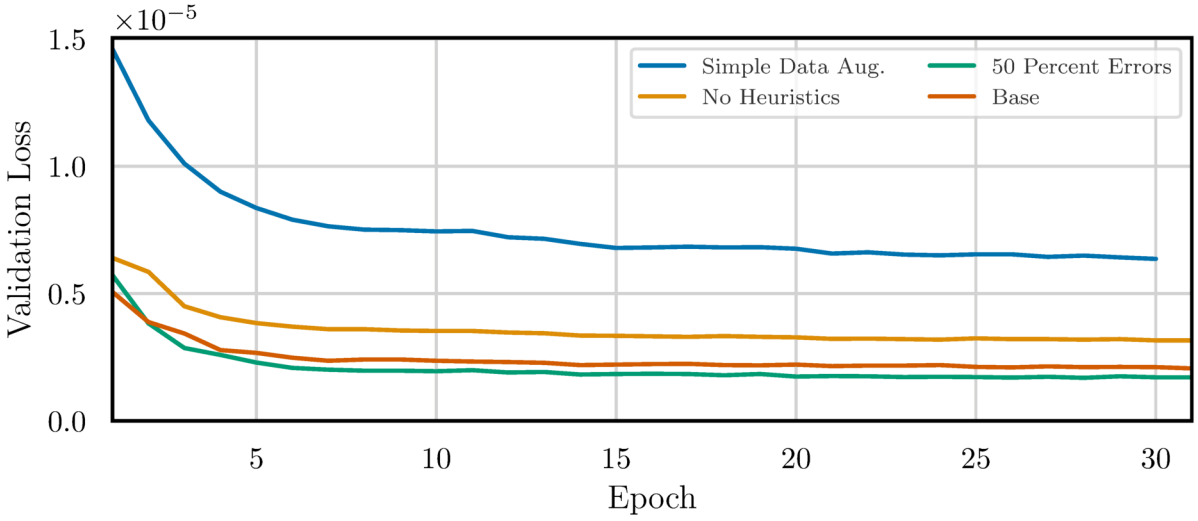


(f) The normalized recall on the validation set of each trial over each epoch, during fine-tuning on real OMR errors.

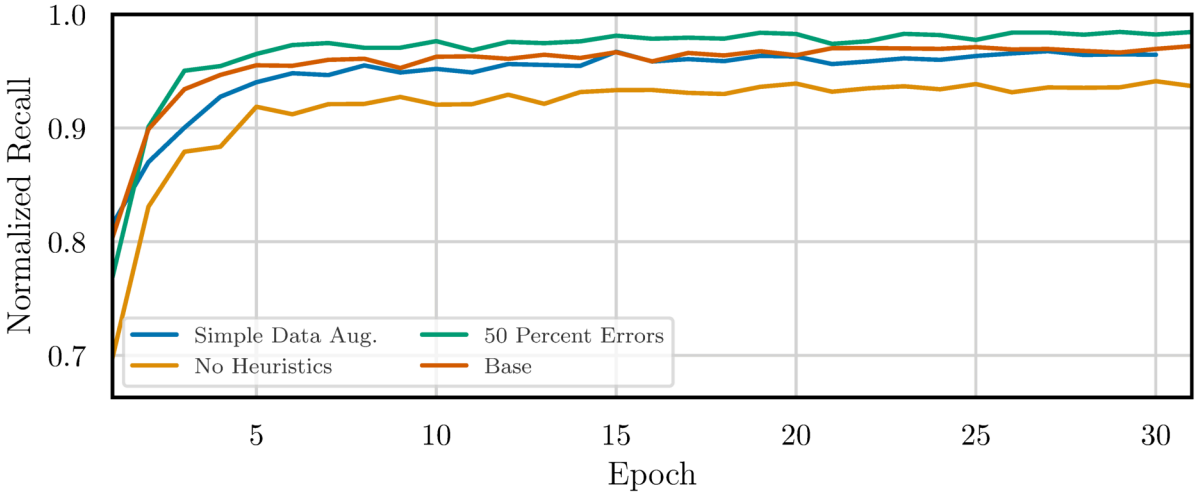
Figure 6.3: Training metrics on the "Architectures" category of trials, each compared against the "Base" trial, which uses the LSTUT model.



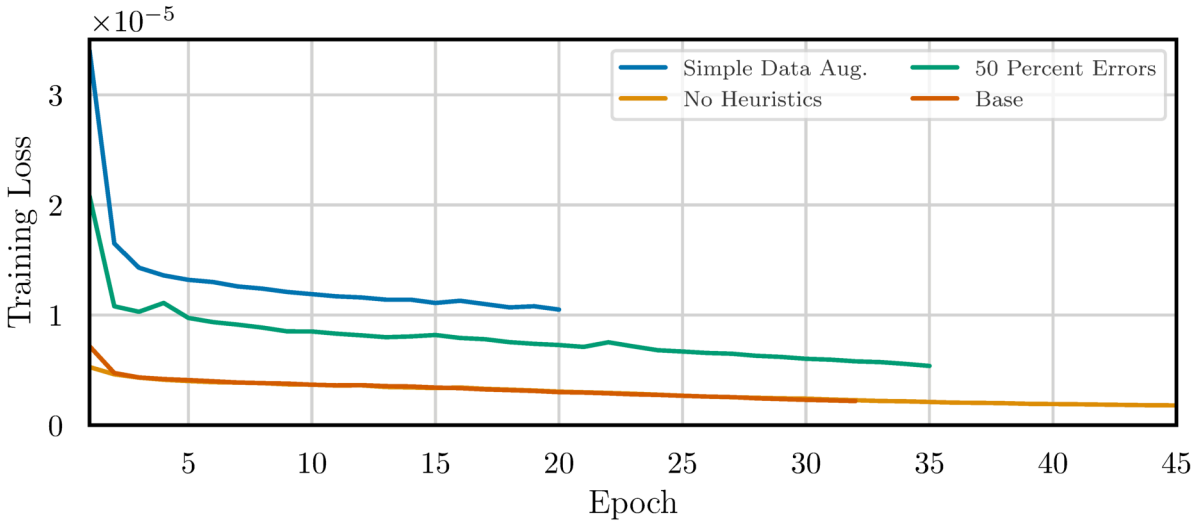
(a) The training loss of each trial over each epoch, during pre-training on synthetic errors.



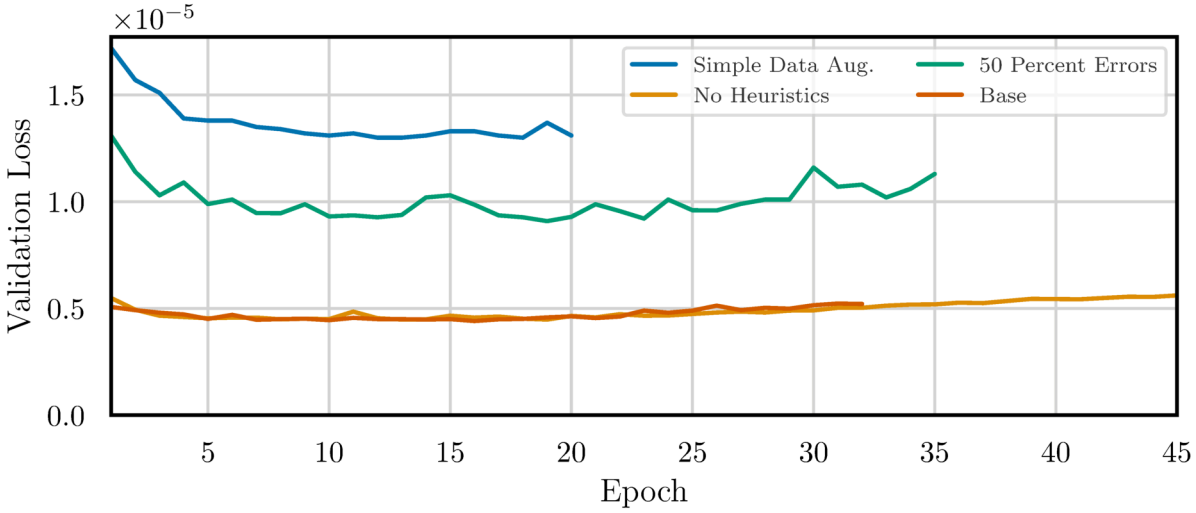
(b) The loss on the validation set on each trial over each epoch, during pre-training on synthetic errors.



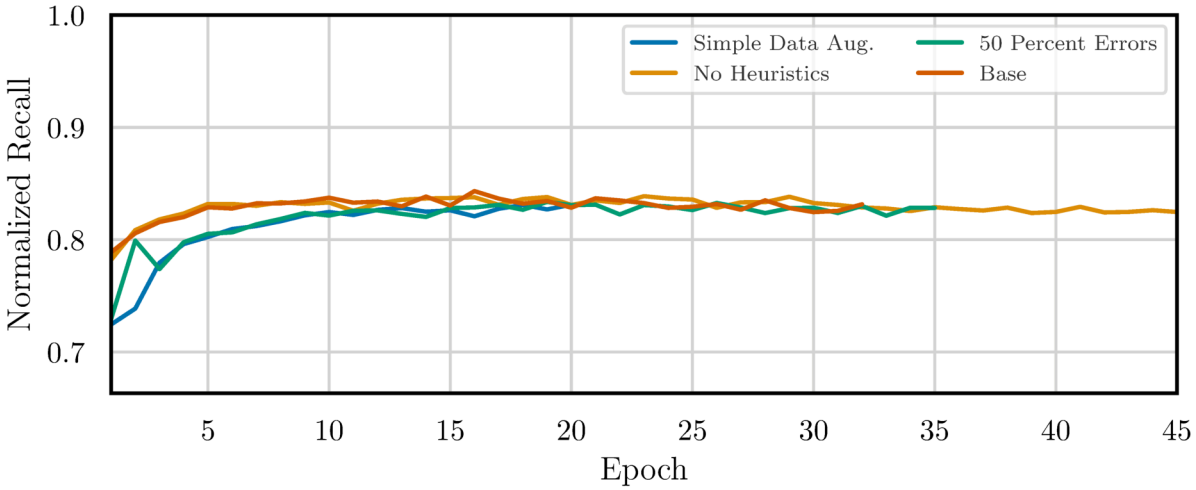
(c) The normalized recall on the validation set of each trial over each epoch, during pre-training on synthetic errors.



(d) The training loss of each trial over each epoch, during fine-tuning on real OMR errors.

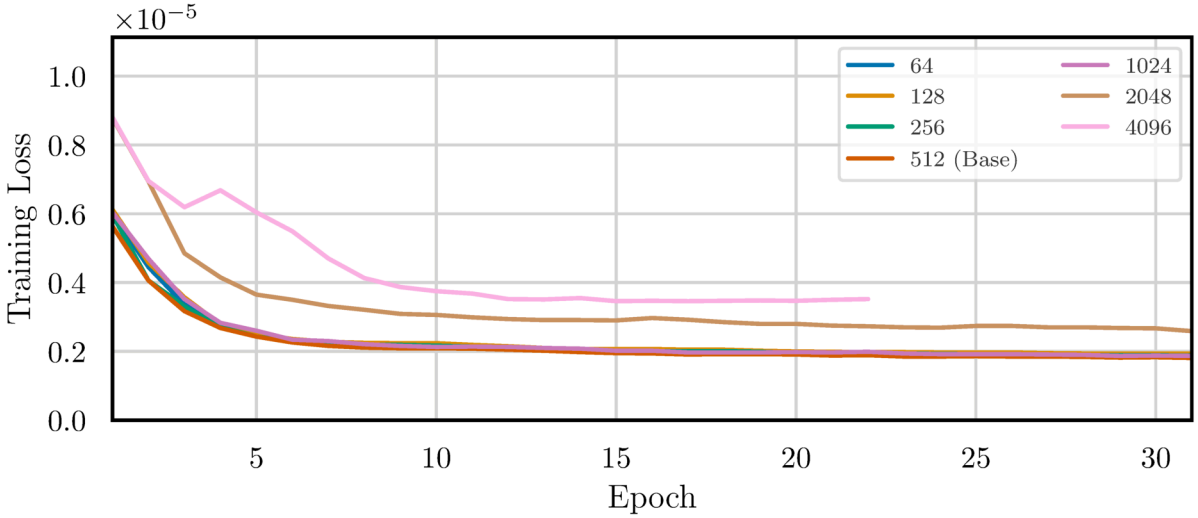


(e) The loss on the validation set on each trial over each epoch, during fine-tuning on real OMR errors.

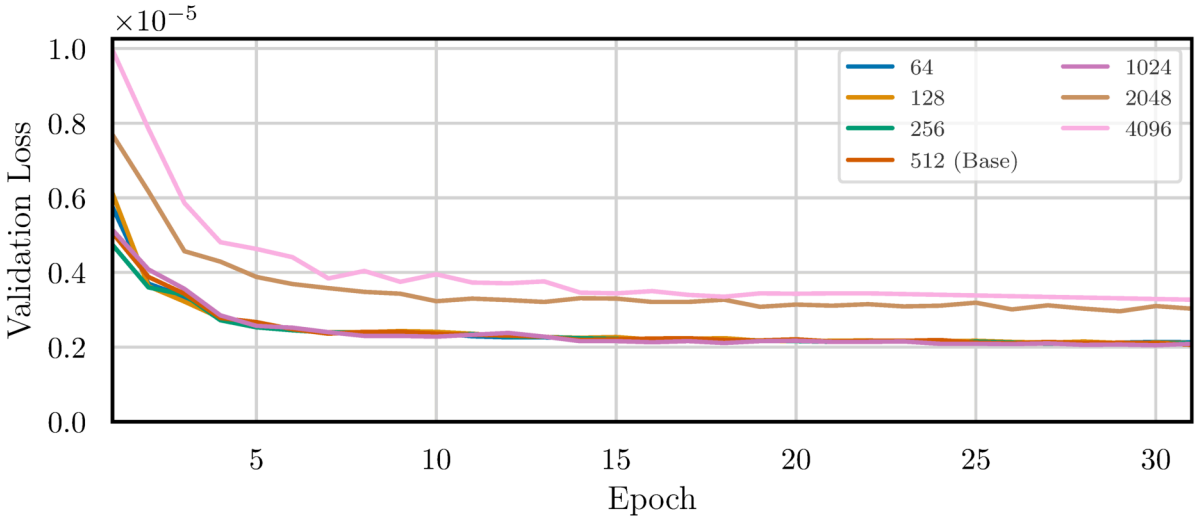


(f) The normalized recall on the validation set of each trial over each epoch, during fine-tuning on real OMR errors.

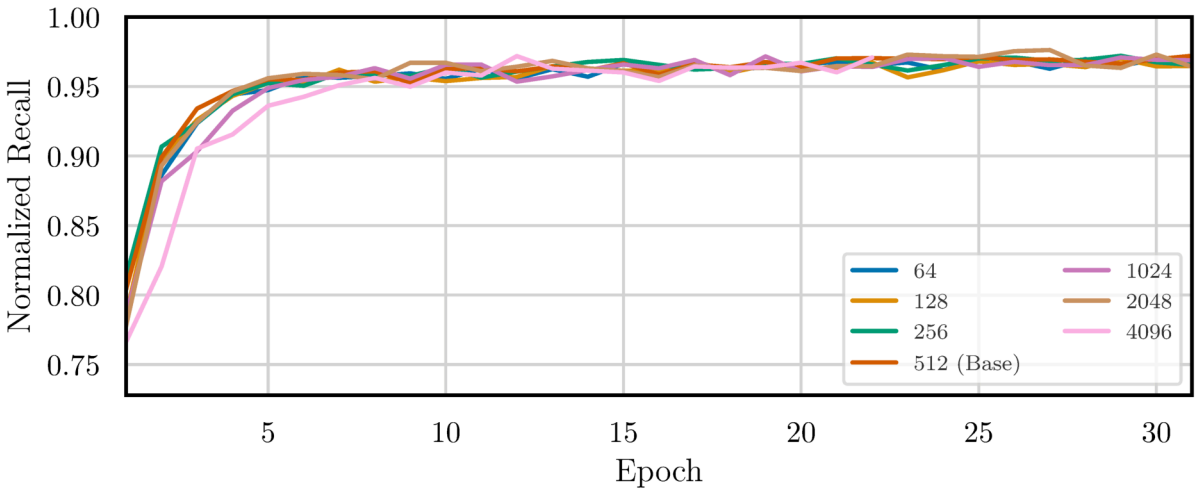
Figure 6.4: Training metrics on the "Data Augmentation" category of trials, each compared against the "Base" model, which uses full probability-based data augmentation with heuristics.



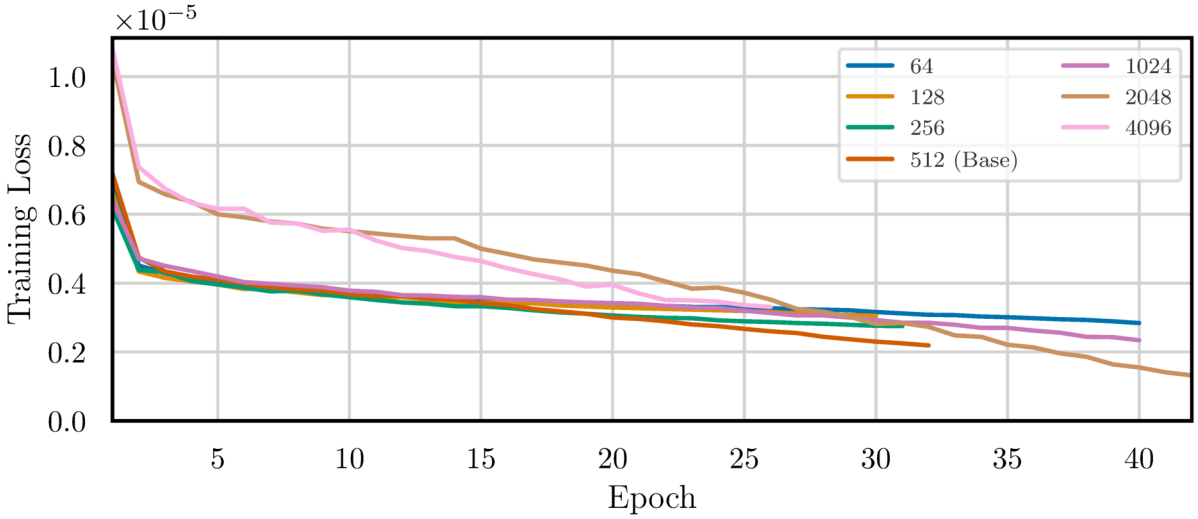
(a) The training loss of each trial over each epoch, during pre-training on synthetic errors.



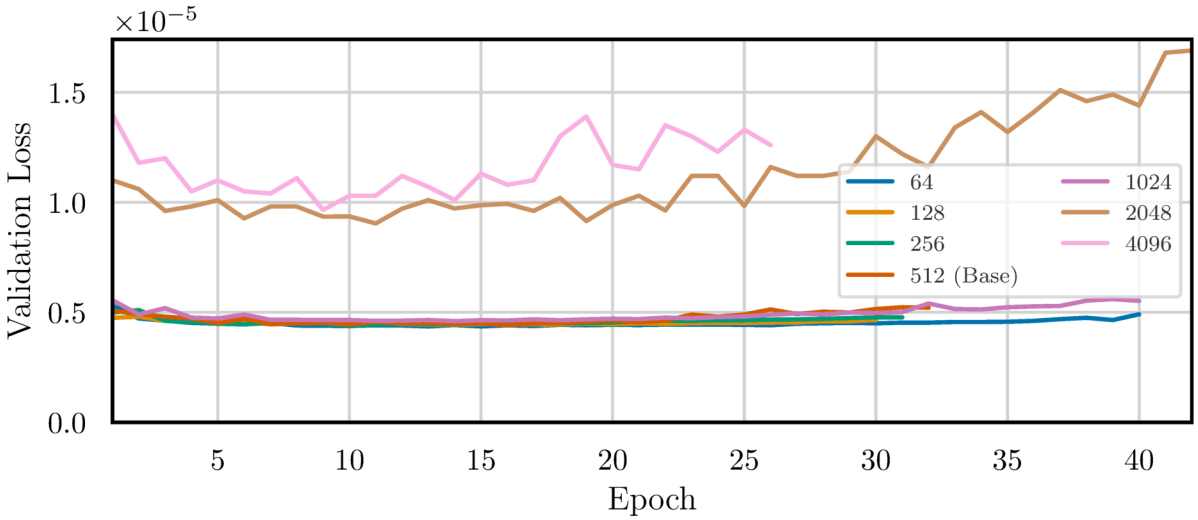
(b) The loss on the validation set on each trial over each epoch, during pre-training on synthetic errors.



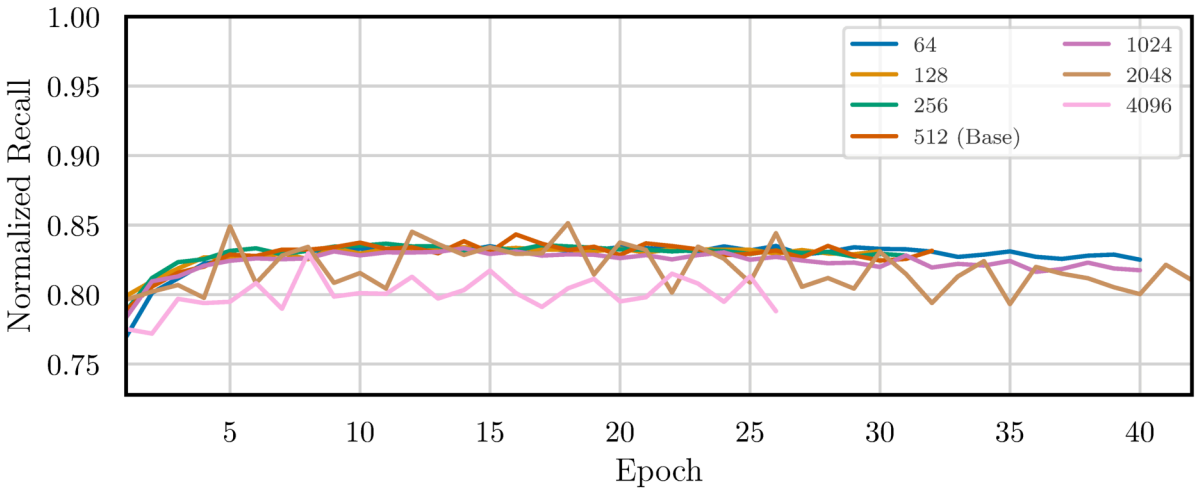
(c) The normalized recall on the validation set of each trial over each epoch, during pre-training on synthetic errors.



(d) The training loss of each trial over each epoch, during fine-tuning on real OMR errors.

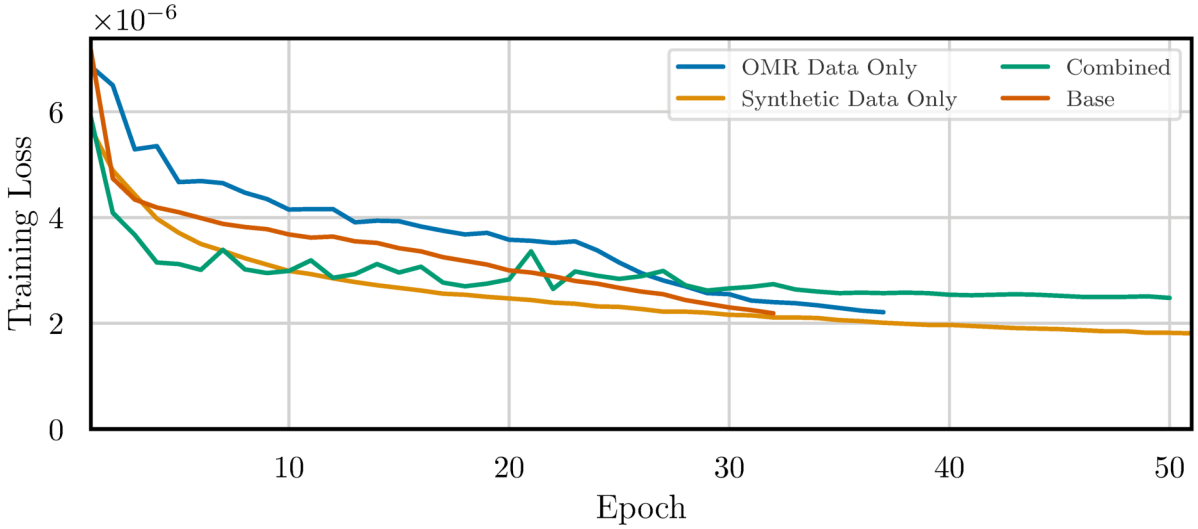


(e) The loss on the validation set on each trial over each epoch, during fine-tuning on real OMR errors.

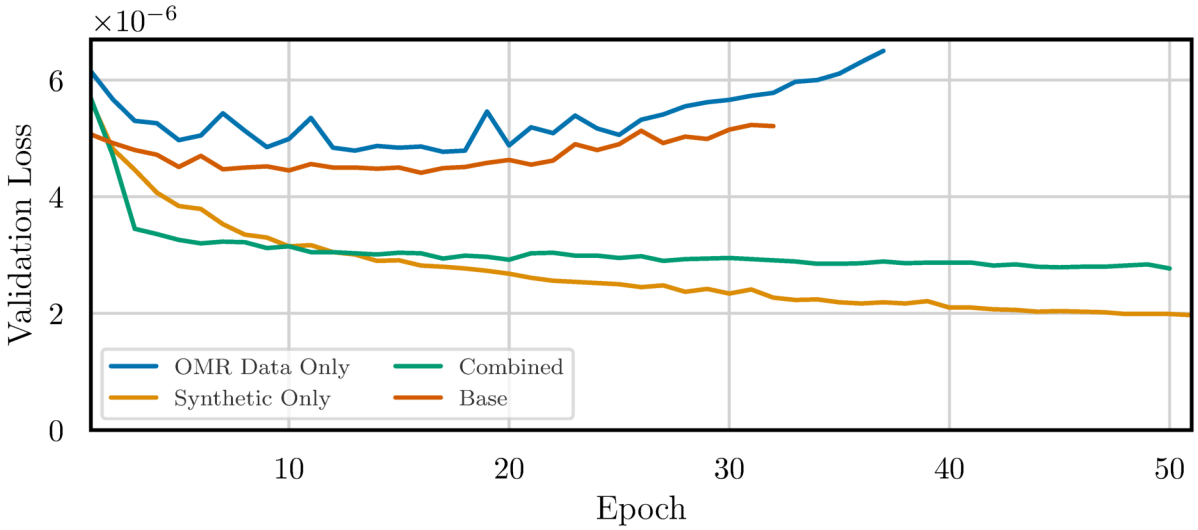


(f) The normalized recall on the validation set of each trial over each epoch, during fine-tuning on real OMR errors.

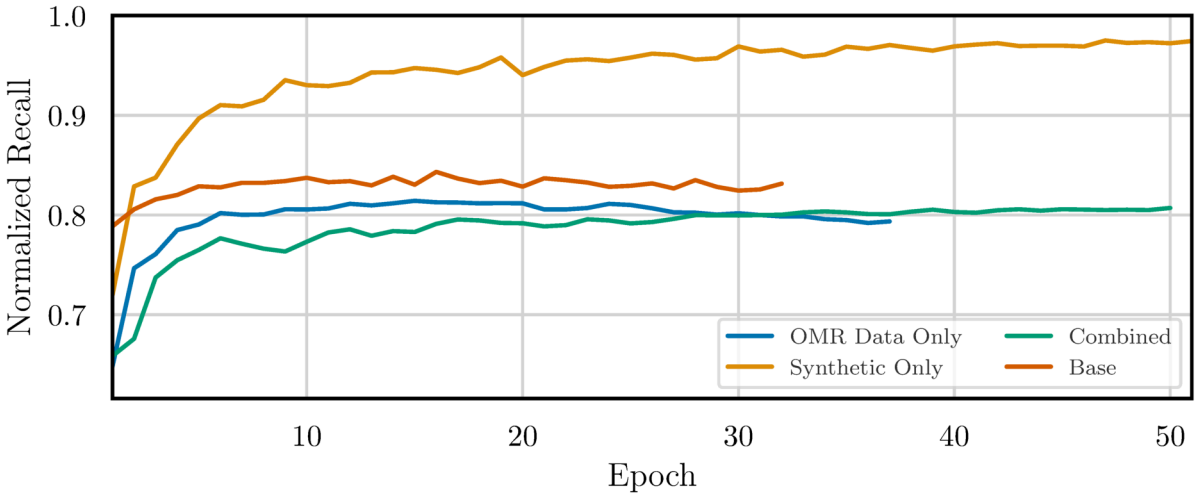
Figure 6.5: Training metrics on the "Sequence Length" category of trials, each compared against the "Base" model, which uses a Sequence Length of 512.



(a) The training loss of each trial over each epoch.



(b) The loss on the validation set on each trial over each epoch.



(c) The normalized recall on the validation set of each trial over each epoch.

Figure 6.6: Training metrics on the "No Fine-Tuning" category of trials, each compared against the "Base" model, which does not use any pre-training step. The "Base" model metrics shown are those calculated during the fine-tuning step, after pre-training has completed.

6.1.2.2 Discussion on Effects of Different Sequence Lengths

From these trials, it appears that changing the input sequence length has relatively little impact on the performance of the model. A sequence length of between 128 and 512 tokens slightly increases the performance of the model on the test set, but this effect is small and could entirely result from how I have changed the batch size along with the input sequence length in these trials. High sequence length corresponds to the low batch sizes under this scheme, and low batch sizes are known to lead to a more slower training process as the gradient updates tend to be noisier. This is evidenced by the loss curves in Figure 6.5, where the trials with the highest sequence lengths converge erratically.

However, another effect of large batch sizes is to cause a model to not generalize as well on test data, as the noise has a normalizing effect on the training process (See the discussion of stochastic gradient descent in Section 4.2.3 for details). These two competing effects mean that it is difficult to gauge how much of the small change in performance between different sequence lengths seen in Tables 6.3 and 6.4 is due to the amount of information available to the model and how much is due to differences in the training process.

6.1.2.3 Discussion on Effects of Data Augmentation Methods

Some of the trials that modify the data augmentation procedures show significantly different results from the "Base" trial. The worst-performing trial in this category was the one that trains only on synthetic errors. Combining the natural and synthetic errors instead of fine-tuning on the natural errors showed a marked decrease in all performance metrics, as did training only on natural errors and using no data augmentation whatsoever. This confirms, as hypothesized in Section 1.2, that the use of synthetic errors in the training process is beneficial to the error detector. However, only using synthetic errors produced a model that performs poorly, so an amount of natural OMR errors must still be necessary to train the model.

Other parameters appear to make little difference, however. Surprisingly, the type of augmentation used to create the synthetic data has little to no effect on the performance of the trained model. Even using randomly generated errors had nearly the same effect as my scheme to generate errors that are statistically similar to natural OMR errors. Furthermore, the heuristics introduced to simulate the clumping together of natural OMR errors appeared to have no effect on the performance of the model at all.

6.1.2.4 Discussion of Results on Other Test Sets

After being trained to completion, in addition to being tested on the test set of OMR data all trained models were evaluated on two other sets of data: A small collection of five of Mendelssohn's string quartets from the Mendelssohn String Quartet Dataset that had been partially corrected by humans, and a segment of the large dataset for generating synthetic errors. Neither of these datasets were shown to the model at any point in the training process. Some results from these tests are shown in Table 6.5.

All models performed equally poorly on the partially corrected OMR output. In the partially corrected data, around 4% of the tokens were errors, so a precision of 0.04 was trivially achievable by predicting all tokens to be errors. The relatively high value for R_{norm} here mainly reflects the models' effectiveness in identifying correct tokens as non-erroneous. In contrast, the AP metric, which solely quantifies the ranking of positively labeled tokens, was significantly worse. On synthetic data, all models performed better than they did on the OMR test data. Keep in mind that the evaluation shown in Table 6.5 occured *after* the models were fine-tuned on the small dataset of OMR data (except for the "Synthetic only" trial which, as expected, performs very well on Synthetic data). This could signal that the synthetic errors are, in a sense, easier to identify than the natural OMR errors are, even for the trial that was trained only on natural errors and saw no synthetic errors at all.

6.1.3 Estimated Reduction in Correction Times

The goal of this error detection system is to reduce the time it takes to manually correct scores after OMR processing. In order to provide a detailed analysis of the error detector's performance and its potential time-saving benefits, this section focuses on statistics for individual string quartet movements from the test set, all containing real OMR errors. Table 6.6 includes a selection of these movements, chosen to ensure that each composer in the test set is represented at least once. The table shows statistics on the length of each piece, the number of agnostic tokens representing each piece, and the distribution of errors within each piece as defined by the Affine Needleman-Wunsch (ANW) algorithm (see Section 5.2.4). The measure count represents the total number of individual measures notated, counted separately on each staff, and not the length of the piece in measures. The OMR process sometimes misidentifies barlines, so each part of each piece may have a different number of measures; this is why the total number of measures in each piece is not always a multiple of four. The test set showcases a broad range of OMR

output quality. The percentage of errors in each piece varies significantly, with some having as low as 7% erroneous tokens, while others have nearly half their content marked as errors.

Table 6.7 provides insights into the performance of the error detector on specific movements from the test set, as analyzed using the LSTUT architecture from the "Base" model. The table includes key metrics relevant to the error detection process and estimates the potential time savings during the OMR correction process. The columns **Prop. Erroneous Tokens** and **Prop. Tokens Marked** correspond to the variables c and e in Equation 3.6, and indicate the actual proportion of errors in the score and the proportion of the score marked as erroneous by the detector. Equation 3.6 also requires a value p, which represents the average proportion of time spent in the correction process spent only on comparing two measures. The rightmost two columns of this table contain estimates for what proportion of the total correction time could be saved, based on an optimistic ($p = 0.59$) and a conservative ($p = 0.24$) estimate for the value of p.

Scores with fewer errors, like Fanny Hensel's String Quartet No. 1, can benefit significantly from the error detector even at low precision; Equation 3.6 predicts substantial time savings on these pieces due to their sparse distribution of errors. Conversely, in pieces with a higher density of errors, such as Haydn's String Quartet No. 36, the detector contributes less to time savings despite higher precision. This is because during the correction process a larger portion of time must be spent correcting numerous errors rather than searching for them. The detector's effectiveness in reducing correction time is influenced by the balance between its precision (the accuracy of its error identifications) and the overall volume of errors in the score. Lower precision can be offset by a lower volume of errors, and higher precision is more beneficial in denser error environments.

Table 6.6: A table showing statistics on the length, number of tokens, and distribution of errors in each piece in the test set of real OMR errors. Not all the pieces in the test set are shown, for lack of space.

String Quartet	**Opus or Catalogue #**	**Length in Measures**	**Total Tokens**	**Total Errors**	**Measures w/Errors**	**Prop. Erroneous Tokens**	**Prop. Measures w/Errors**	**Tokens per Measure**	**Errors per Measure**
Beethoven, Große Fuge	Op. 133	2993	24064	7530	1875	0.31	0.63	8.04	2.5
Borodin, No. 2, Mvt. 1	IAB 14	1122	8192	2019	619	0.25	0.55	7.30	1.8
Borodin, No. 2, Mvt. 3	IAB 14	699	6144	2692	569	0.44	0.81	8.79	3.9
Carreño, No. 1, Mvt. 1	ITC 12	554	5618	813	280	0.14	0.51	10.14	1.5
Carreño, No. 1, Mvt. 2	ITC 12	444	4608	579	200	0.13	0.45	10.38	1.3
Carreño, No. 1, Mvt. 4	ITC 12	556	6645	609	227	0.09	0.41	11.95	1.1
Grieg, No. 1, Mvt. 1	Op. 27	2410	23040	3494	1091	0.15	0.45	9.56	1.4
Haydn, No. 1	Op. 1-i	1136	8192	2870	855	0.35	0.75	7.21	2.5
Haydn, No. 36	Op. 20-vi	1524	15324	5836	1082	0.38	0.71	10.06	3.8
Haydn, No. 53	Op. 64-vi	1916	16879	3034	970	0.18	0.51	8.81	1.6
Hensel, No. 1, Mvt. 1	HelH. 277	308	1987	146	70	0.07	0.23	6.47	0.5
Hensel, No. 1, Mvt. 2	HelH. 277	696	5568	548	216	0.10	0.31	8.00	0.8
Hensel, No. 1, Mvt. 4	HelH. 277	996	9706	676	272	0.07	0.27	9.72	0.7
Mendelssohn, No. 2, Mvt. 4	Op. 13	1508	11713	4597	1028	0.39	0.68	7.77	3.0
Mendelssohn, No. 3, Mvt. 2	Op. 44-i	859	4608	948	334	0.21	0.39	5.36	1.1
Mendelssohn, No. 4, Mvt. 2	Op. 44-ii	843	7655	1583	460	0.21	0.55	9.08	1.9
Mendelssohn, No. 5, Mvt. 3	Op. 44-iii	508	4096	1855	430	0.45	0.85	8.06	3.7
Schubert, No. 14, Mvt. 2	D. 810	804	9686	987	378	0.10	0.47	12.05	1.2
Schubert, No. 14, Mvt. 3	D. 810	536	6100	985	255	0.16	0.48	11.38	1.8
Schubert, Quartettsatz	D. 103	1227	9728	955	332	0.10	0.27	7.93	0.8
Schumann, No. 1, Mvt. 2	Op. 41-i	829	7650	1766	551	0.23	0.66	9.23	2.1
Schumann, No. 1, Mvt. 3	Op. 41-i	331	3576	803	201	0.22	0.61	10.80	2.4
Average		1083.9	9763.6	2232.9	602.2	0.22	0.54	9.22	2.0
Standard Deviation		645.5	5656.1	1814.3	415.8	0.12	0.17	1.69	1.0

Table 6.7: A table showing the performance of the error detector on each piece in the test set of real OMR errors. These tests use the "Base" LSTUT model and a sequence length of 512. The rightmost two columns use Equation 3.6 to calculate an estimate for how much the error detector would reduce the amount of time needed to correct that piece, and the two values for p used were upper and lower bounds calculated in Section 3.4.2. Not all the pieces in the test set are shown, for lack of space.

	At 0.9 Recall						Estimated Reduction in Correction Time, with	
String Quartet	**Prop. Erroneous Tokens c**	**Prop. Tokens Marked e**	**Prop. Measures w/Errors**	**Marked Tokens per Measure**	**Precision**	**R_{norm}**	**p=0.24**	**p=0.59**
Beethoven, Große Fuge	0.31	0.59	0.63	4.73	0.48	0.86	0.14	0.23
Borodin, No. 2, Mvt. 1	0.25	0.63	0.55	4.61	0.35	0.82	0.16	0.24
Borodin, No. 2, Mvt. 3	0.44	0.69	0.81	6.05	0.57	0.83	0.07	0.13
Carreño, No. 1, Mvt. 1	0.14	0.86	0.51	8.74	0.15	0.67	0.08	0.11
Carreño, No. 1, Mvt. 2	0.13	0.82	0.45	8.49	0.14	0.70	0.11	0.15
Carreño, No. 1, Mvt. 4	0.09	0.58	0.41	6.90	0.14	0.80	0.30	0.36
Grieg, No. 1, Mvt. 1	0.15	0.40	0.45	3.84	0.34	0.90	0.34	0.46
Haydn, No. 1	0.35	0.54	0.75	3.87	0.59	0.88	0.14	0.24
Haydn, No. 36	0.38	0.50	0.71	5.02	0.69	0.91	0.14	0.25
Haydn, No. 53	0.18	0.49	0.51	4.28	0.33	0.88	0.27	0.38
Hensel, No. 1, Mvt. 1	0.07	0.68	0.23	4.42	0.09	0.75	0.24	0.28
Hensel, No. 1, Mvt. 2	0.10	0.51	0.31	4.08	0.17	0.85	0.34	0.41
Hensel, No. 1, Mvt. 4	0.07	0.39	0.27	3.77	0.16	0.89	0.47	0.54
Mendelssohn, No. 2, Mvt. 4	0.39	0.60	0.68	4.64	0.59	0.86	0.11	0.19
Mendelssohn, No. 3, Mvt. 2	0.21	0.44	0.39	2.36	0.42	0.90	0.27	0.39
Mendelssohn, No. 4, Mvt. 2	0.21	0.49	0.55	4.46	0.38	0.87	0.24	0.35
Mendelssohn, No. 5, Mvt. 3	0.45	0.66	0.85	5.29	0.62	0.85	0.08	0.14
Schubert, No. 14, Mvt. 2	0.10	0.60	0.47	7.24	0.15	0.80	0.27	0.33
Schubert, No. 14, Mvt. 3	0.16	0.58	0.48	6.56	0.25	0.84	0.23	0.32
Schubert, Quartettsatz	0.10	0.53	0.27	4.20	0.17	0.87	0.32	0.40
Schumann, No. 1, Mvt. 2	0.23	0.56	0.66	5.15	0.37	0.86	0.20	0.29
Schumann, No. 1, Mvt. 4	0.13	0.42	0.40	4.12	0.27	0.90	0.36	0.46
Average	0.22	0.58	0.54	5.33	0.35	0.84	0.21	0.29
Standard Deviation	0.12	0.11	0.17	1.50	0.18	0.06	0.10	0.11

Figure 6.7 presents a graphical analysis showing the relationship between the error proportion in each score and the predicted time savings due to the use of the error detector. This analysis encompasses all movements in the test set, not just those highlighted in Tables 6.6 and 6.7. The graph reveals a discernible negative correlation: as the proportion of errors in a score decreases, the estimated time savings offered by the error detector increase. For significant time savings (over 25%), the OMR process should incorrectly predict less than 20% of the score.

Of particular interest are three outliers located in the lower left part of the graph, corresponding to three movements from Teresa Carreño's String Quartet No. 1. The first movement stands out with the lowest estimated time savings in the entire test set, despite a relatively low proportion of erroneous tokens (the value of the parameter c). Upon closer examination, I observed that the discrepancies are largely due to notational differences between the original scans and their ground-truth transcriptions provided by the OSSQ. Specifically, these discrepancies often involve the notation of two-note tremolos, as illustrated in Figure 6.8. The OMR process by PhotoScore, attempting to replicate the notation of the original score, fully notates these tremolos. However, the error detector does not identify these as errors, since there is no signal to tell it that anything is amiss with the musical material. This mismatch between the ground-truth and the OMR output leads to lower evaluated accuracy of the error detector. Consequently, to achieve a high recall, the detector marks a vast portion of the score as erroneous. This issue, where the notation differs between the ground-truth transcription and the original score, and its impact on the error detector's performance, appears again in the following section, and will be discussed as a data sanitation problem in Section 7.3.4.

6.2 Example Outputs

In this section I provide lengthy examples of the outputs of the error detector on scores from the test and validation sets of the small dataset containing natural OMR errors. Each example is accompanied by a reproduction of the score image that was fed into PhotoScore to create it.

On each of the scores that follow, symbols are colored according to how they were evaluated by the error detection model and by whether or not they are marked as errors in the ground truth. The "Base" model is used for all examples (See Table 6.1). The detection threshold is set to yield a recall score of 0.90 on the validation set, which implies that approximately 10% of errors in each score are expected to be missed. Symbols are color-coded based on their evaluation

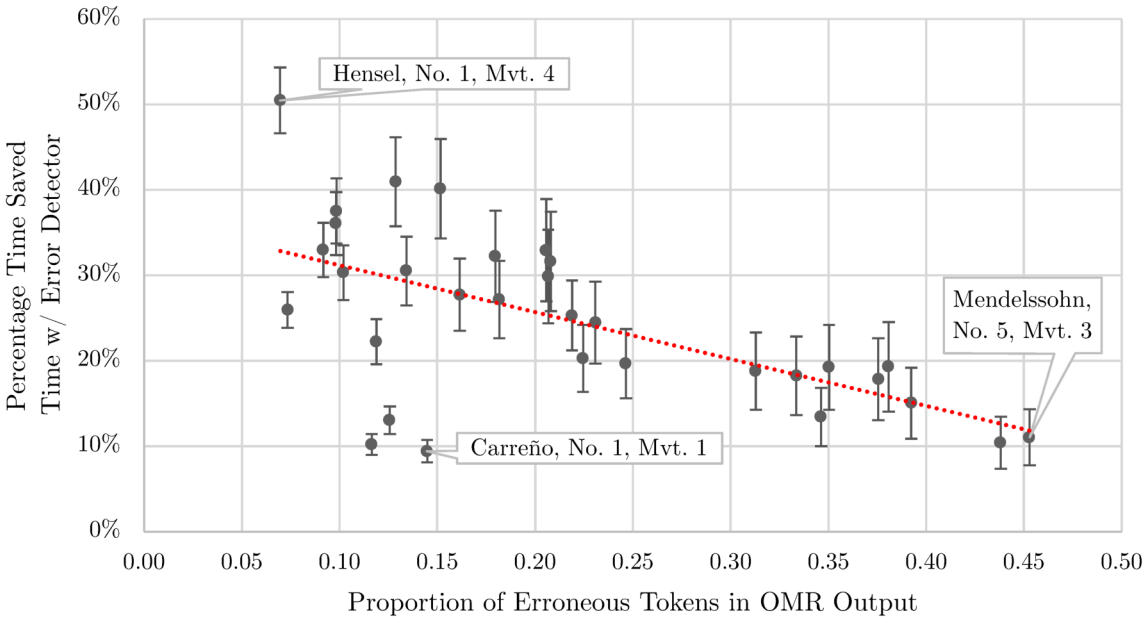
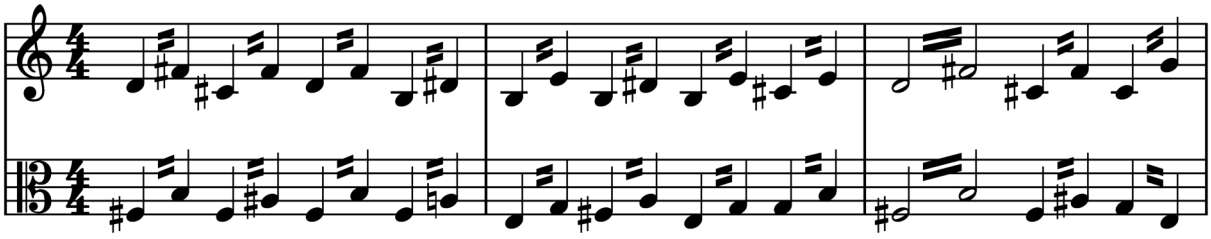
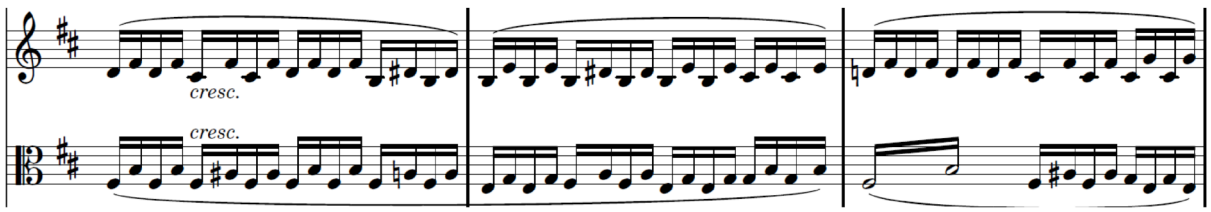


Figure 6.7: A scatter plot showing the relationship between the proportion of the score containing errors and the estimation of the percentage of time the error detector would save on each piece in the test set. The error bars correspond to the upper and lower bounds for p, representing an optimistic and pessimistic estimation for how much time the error detector could save. The line of best fit is shown in as a dotted orange line ($R^2 = 0.38$).



(a) The excerpt as transcribed by the OSSQ.



(b) How the excerpt appears in the original scan that was transcribed.

Figure 6.8: Teresa Carreño's String Quartet No. 1, Mvt. 1, mm. 11–13, 2nd violin and viola parts. Figure (a) shows how the excerpt was transcribed by the OSSQ, while (b) shows how it was originally notated.

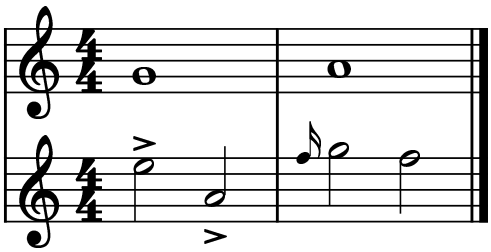
by the error detection model and their status in the ground truth. True positives are colored yellow, false positives red, false negatives blue, and true negatives remain black (refer to Section 4.1.1 for term definitions). When viewing the annotated scores, recall that the error detector aims to eliminate all blue-marked symbols (false negatives) while minimizing the number of red-marked symbols (false positives).

6.2.1 Interpreting these Examples

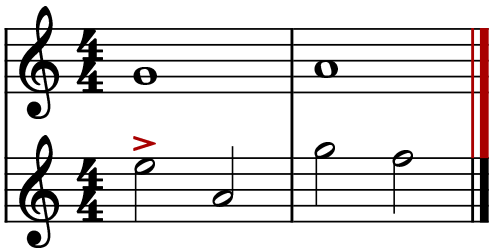
The aforementioned figures were generated automatically, from the MusicXML files created by the inference process described in Section 5.5. Due to technical restrictions involving the technologies used, the semantic-to-agnostic alignment and coloring processes are necessarily imperfect, and the examples rendered below must be interpreted with these imperfections in mind. Furthermore, using the predictions of the error correction model to modify the corresponding symbols in a semantically encoded file requires an amount of musical interpretation of the score itself, and so there are many edge cases where it is not clear what the "correct" behavior of this system should be. The problem of creating a robust, interpretable visualization system for error detection results would likely constitute another entire research project (See Section 7.3 for a discussion of possible future work). Here I describe the issues with the current system and how they affect how the colored examples must be interpreted.

My definition of error, using the ANW algorithm (see Section 2.4.3), permits numerous situations where notes are marked as erroneous for reasons that are not musically intuitive. Often, this is because the ANW algorithm must make trade-offs between local accuracy and globally minimizing the total number of operations needed to align the two sequences. A token may be aligned with a gap in the other sequence despite having an obvious correct counterpart if that one error allows the sequences to have a lower-cost global alignment. See Table 2.3 for an example of this phenomenon and how it interacts with affine gap penalties. Additionally, notes can be marked as erroneous on the basis of notational changes that would not affect performance, such as changes in in stem direction and beaming.

Another subtlety of my definition of error is that a token considered correct may still be marked as an error if a subsequent insertion is required in the agnostic encoding to make the entire sequence correct. The issue here is that two glyphs that are close together in the agnostic encoding may be visually far apart in the score. The agnostic encoding scheme serializes scores in a particular order: within a measure, it goes from left to right, and from bottom to top for



(a) A short score.



(b) The score (a) with errors introduced.

Score (a)	Score (b)	Operation
clef.treble	clef.treble	SAME
timeSig.4/4	timeSig.4/4	SAME
whole.noBeam.up.pos2	whole.noBeam.up.pos2	SAME
barline.regular	barline.regular	SAME
clef.treble	clef.treble	SAME
timeSig.4/4	timeSig.4/4	SAME
half.noBeam.down.pos7	half.noBeam.down.pos7	SAME
∧	∧	SAME
articulation.accent.pos6	articulation.accent.pos6	SAME
articulation.accent.pos0	_	INSERT
∧	∧	SAME
half.noBeam.up.pos3	half.noBeam.up.pos3	SAME
barline.regular	barline.regular	SAME
barline.return-to-top	barline.return-to-top	SAME
whole.noBeam.up.pos3	whole.noBeam.up.pos3	SAME
barline.final	barline.final	SAME
acciaccatura.16th.noBeam.up.pos8	_	INSERT
half.noBeam.down.pos9	half.noBeam.down.pos9	SAME
half.noBeam.down.pos8	half.noBeam.down.pos8	SAME
barline.final	barline.final	SAME
barline.return-to-top	barline.return-to-top	SAME

(c) An ANW alignment between the scores (a) and (b), and the operations necessary to turn (b) back into (a).

Figure 6.9: Two scores that, when compared with the ANW algorithm, result in the alignment shown in (c). Under the scheme used to mark errors in OMR output, when a token must be inserted in order to correct the score, the token immediately before the point of insertion is marked as an error. The tokens that would be considered errors under this alignment are colored red both in the notated example (b) and in its corresponding agnostic encoding in (c).

glyphs at the same vertical position. This means that in a vertical stack of symbols on the staff, the top one may be marked as an error due to a necessary insertion at a later point in the staff. In addition, measures are processed sequentially from each staff, starting from the top staff and moving downwards. The last glyph in a measure, which will always be a barline, may be flagged as erroneous if an insertion is required at the beginning of the same measure on the following staff. These two unintuitive phenomena are shown in Figure 6.9.

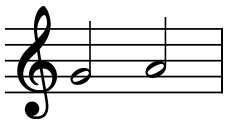
For the purposes of this evaluation all slurs are ignored, and so a misidentified slur in the OMR will never be considered an error by the model nor by the alignment against the ground truth. This is because of issues with how rarely slurs are perfectly detected in PhotoScore on

even slightly degraded score images. See Section 5.2.3 for details.

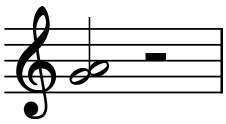
The inference process parses a semantically encoded music file into a `music21` stream (Cuthbert and Ariza 2010), runs the musical material through the trained error detection model, and then uses the results to assign colors to objects in the stream. The resulting colored stream is exported to MusicXML using `music21`'s built-in MusicXML converter, and the resulting MusicXML file is engraved and rendered in vector graphics with the Verovio engraving library (Pugin et al. 2014). A few types of musical glyph cannot be colored with this process. Tuplet markers, for example, are not explicitly defined as attributes or tags in MusicXML, and they cannot be assigned colors in a `music21` Stream; there is no object representing a tuplet marker to modify. Their position on the page must be inferred from the presence of consecutive notes with tuplet durations. Though it is possible to assign barlines, ties, and dynamic markings a color in MusicXML and in a `music21` Stream, Verovio appears not to carry this color information over when importing a MusicXML file.[4] Using the interfaces of modern notation editors, meanwhile, it is possible to manually assign colors to every glyph, including tuplet markers, ties, dynamics, and barlines. After testing the well-established editors I have access to (Finale, MuseScore, and Sibelius), however, I found that when importing a MusicXML file, none of them import the coloring of symbols other than note heads. I am unaware of any method of automatically engraving MusicXML files that respects the colors assigned in the MusicXML file to *all* musical glyphs, so the Verovio solution is what I use here.

When converting a semantically encoded music file into an agnostic format, I maintain a semantic-agnostic alignment, which is a record of the relationship between elements in the semantic file and agnostic tokens. After a prediction for each token is generated, the results of the prediction must be cross-referenced with the original semantic file to assign colors to the appropriate glyphs. The intricate nature of semantically encoded music makes this conversion non-trivial, with the process varying based on the hierarchical organization of each semantic encoding format. In MusicXML, for instance, articulations are not standalone tags but are attributes assigned to notes. Tuplet markers present further difficulties, as detailed earlier. Corrupted MusicXML files introduce challenging scenarios; they might have invalid beaming setups or ties that begin and do not end. These files often result from PhotoScore's OMR

4. Under the hood, Verovio converts any imported MusicXML files into its native Music Encoding Initiative (MEI) format before rendering them graphically. This converter is custom-built for Verovio specifically, and the documentation (`verovio.org/musicxml.html`) warns that it is not necessarily complete, in that some MusicXML attributes and features may not be carried over to the output MEI file. Other extant methods of converting between MusicXML and MEI appear to have similar shortcomings.



(a) A short score.



(b) The score (a) with errors introduced.

Score (a)	Score (b)	Operation
clef.treble	barline.regular	SAME
half.noBeam.up.pos3	half.noBeam.up.pos3	SAME
_	∧	DELETE
half.noBeam.up.pos4	half.noBeam.up.pos4	SAME
_	rest.half	DELETE
barline.regular	barline.regular	SAME

(c) An ANW alignment between the scores (a) and (b), and the operations necessary to turn (b) back into (a).

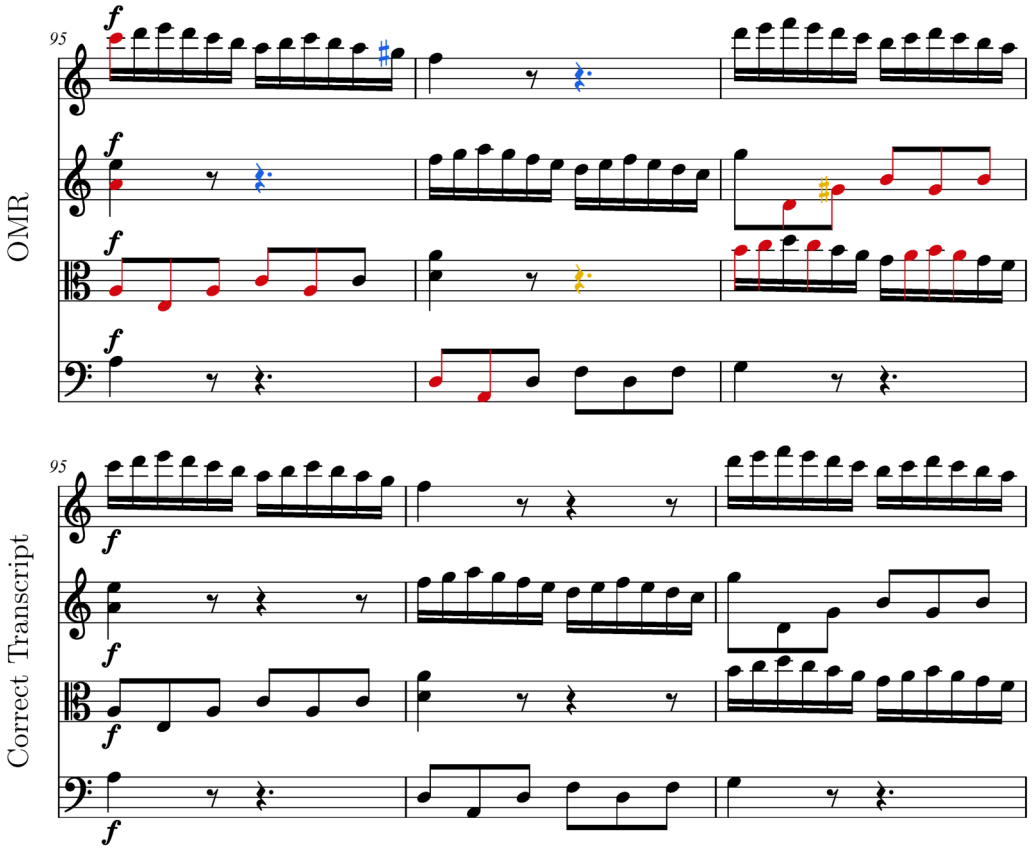
Figure 6.10: Two scores that, when compared with the ANW algorithm, result in the alignment shown in (c). Note that the first error in this example lies on a ∧ token, which is not a symbol that can be colored.

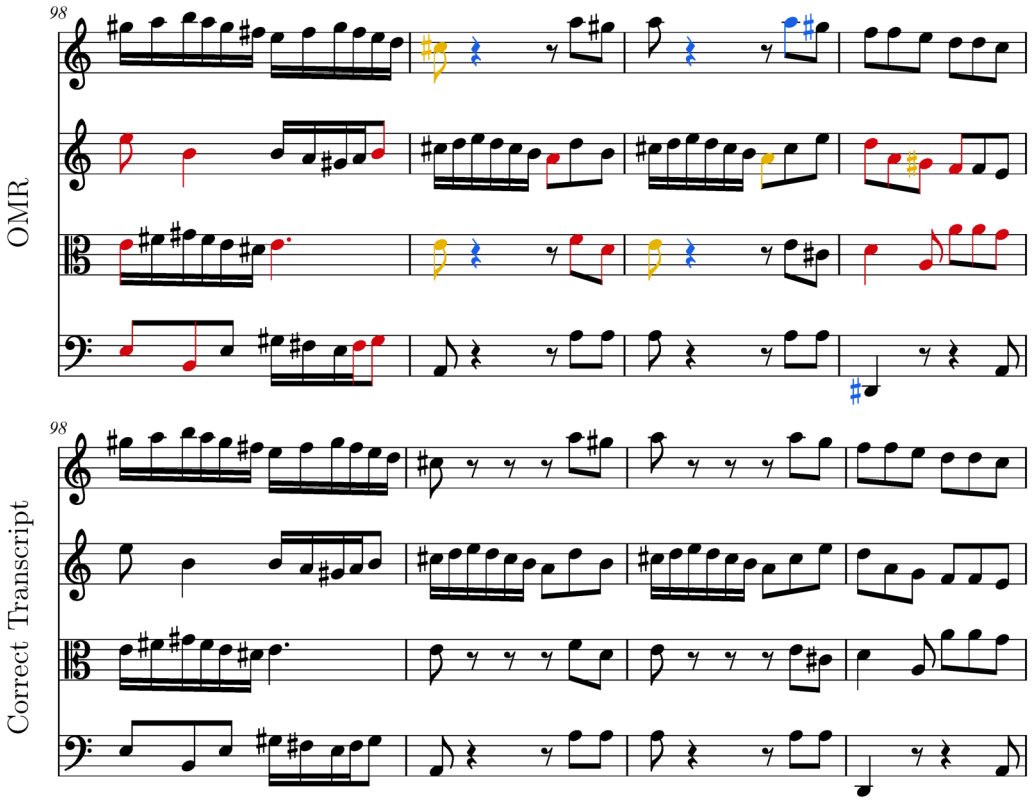
procedure on significantly deteriorated images. Additionally, certain agnostic tokens, like the ∧ token that signifies that the following token is positioned above the preceding one, do not have a direct score symbol counterpart. It is not obvious how to handle the case where a ∧ token is marked as erroneous but the glyphs on either side of it are not. Two scores whose ANW alignment creates this situation are shown in Figure 6.10. As a fallback, when rendering the examples below, I disregard the method's predictions on ∧ tokens entirely.

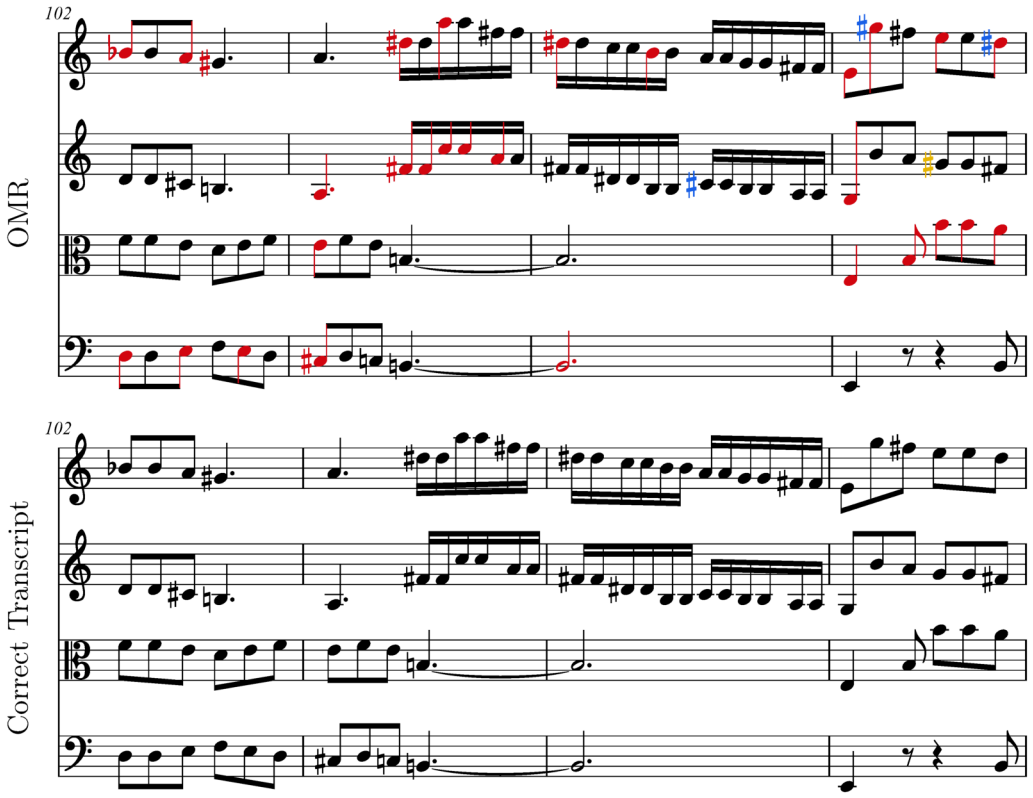
6.2.2 Fanny Hensel - String Quartet in E-flat major

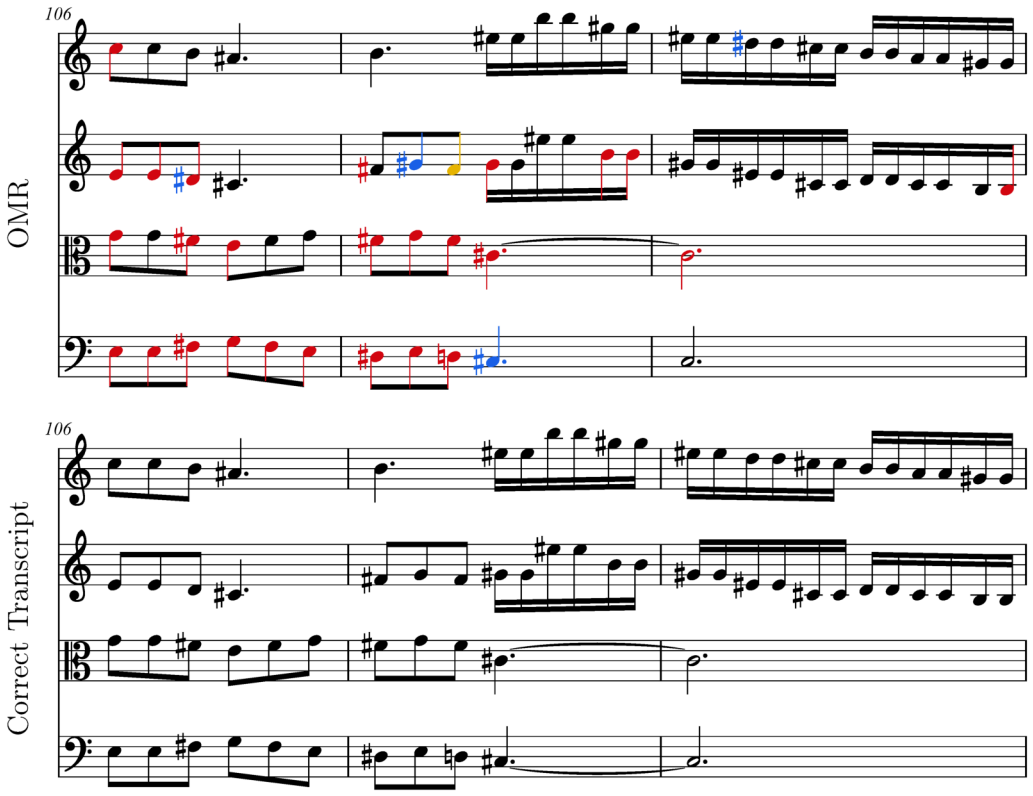
Fanny Hensel Mendelssohn's String Quartet in E-Flat major is a piece that does not use many of the notational phenomena that PhotoScore has a difficult time predicting, such as tremolo, double stops, complicated tuplets, or articulation-heavy, dense passages. In addition, the score I obtained to run through the OMR process is a digital image that had never been printed out and scanned. The OMR output on this string quartet represents the highest-quality possible result from PhotoScore. Figure 6.11 shows an extended excerpt from the quartet with the results of the error detector highlighted alongside the corresponding excerpts from the human-corrected transcript of the piece from the OSSQ. The model performs well overall and though there are some evident false negatives, they tend to occur adjacent to a place where the model has predicted an error.

There are two phenomena in this figure where the Needleman-Wunsch alignment seems to define errors where there should be none, leading to false negatives or true positives. In the first


OMR
95
Correct Transcript
95


OMR
98
Correct Transcript
98


102
OMR
102
Correct Transcript


106
OMR
106
Correct Transcript

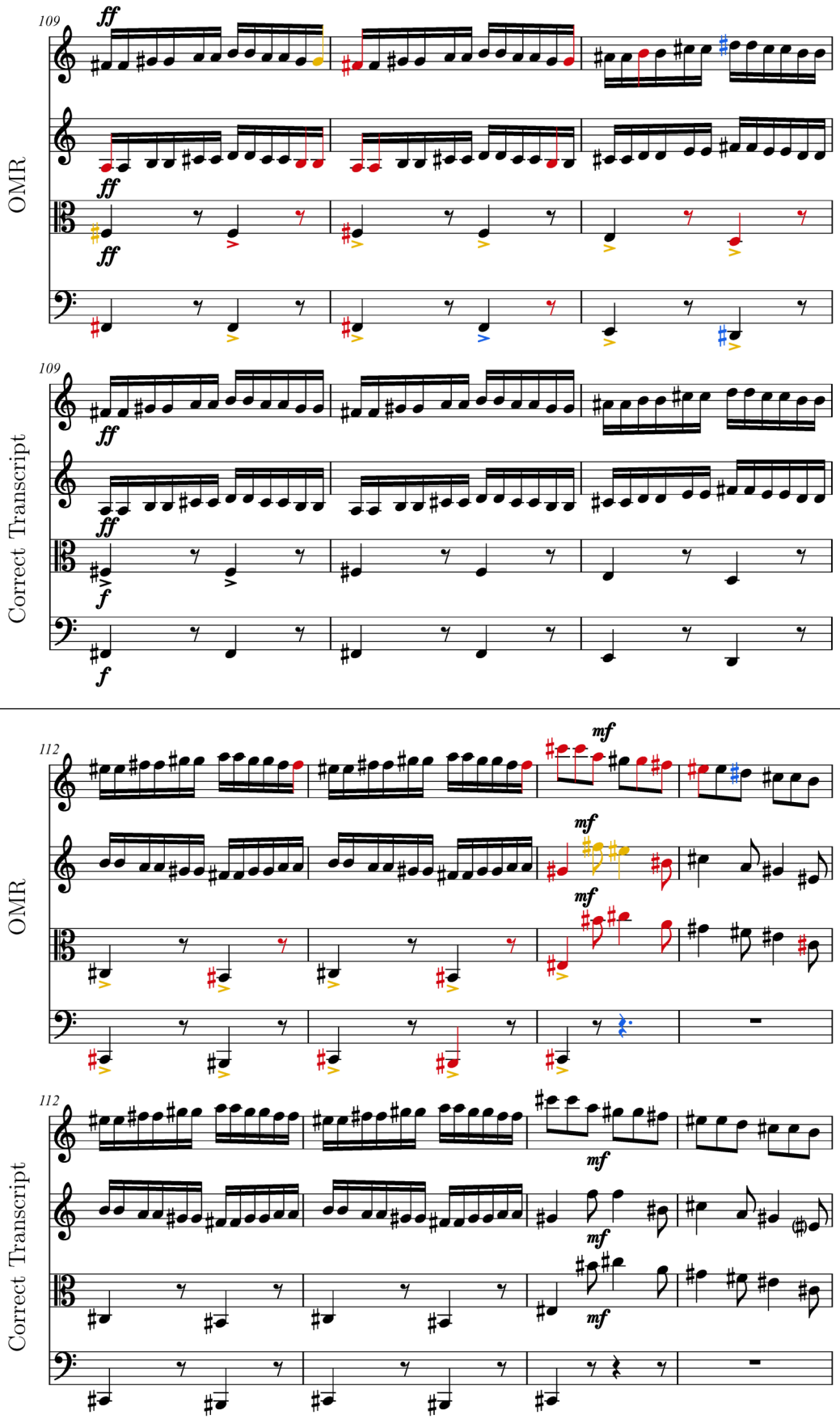


Figure 6.11: Fanny Hensel Mendelssohn's String Quartet in E-Flat major, Mvt. 2, mm. 95–115. The OMR output from PhotoScore and the hand-corrected score are displayed side-by-side. Yellow symbols are correct detections, red symbols are false positives, and blue symbols are false negatives.

few measures of the excerpt, a number of dotted quarter rests have been marked as erroneous in the OMR output despite those quarter rests being present in the original image of the score. In the second system, in measure 99, the first violin and viola have eighth notes that appear to be identical between the OMR and the transcript, but this eighth note is marked as an error; this is because the ANW algorithm has prescribed the addition of eighth rests immediately following that note in order to correct the score, so it is marked as an error as well. In addition, in the second half of the excerpt, the viola and cello parts both have repeated sections of accented quarter notes interspersed with eighth rests. Many of these accents are marked as errors despite also being present in the scan of the score.

After some investigation, I found that this is not the result of a mistake by PhotoScore itself, which correctly recognizes the dotted quarter notes and accent marks in the original score and encodes them properly in its MusicXML output. This is, in fact, an "error" in the human-corrected transcript that I am using as ground truth to align against the version with OMR errors. The human-corrected transcript of this string quartet did not transcribe all of the dotted quarter rests as dotted quarter rests; most of them were transcribed as a quarter rest followed by an eighth rest. PhotoScore, meanwhile, reproduced the dotted quarter rests correctly. In the section containing accent marks on quarter notes, the transcriber only placed them on the quarter notes in the first measure. The OMR'ed score, however, contains every accent mark present in the original engraving. On aligning the two scores together, these conflicts lead to the appearance of errors.

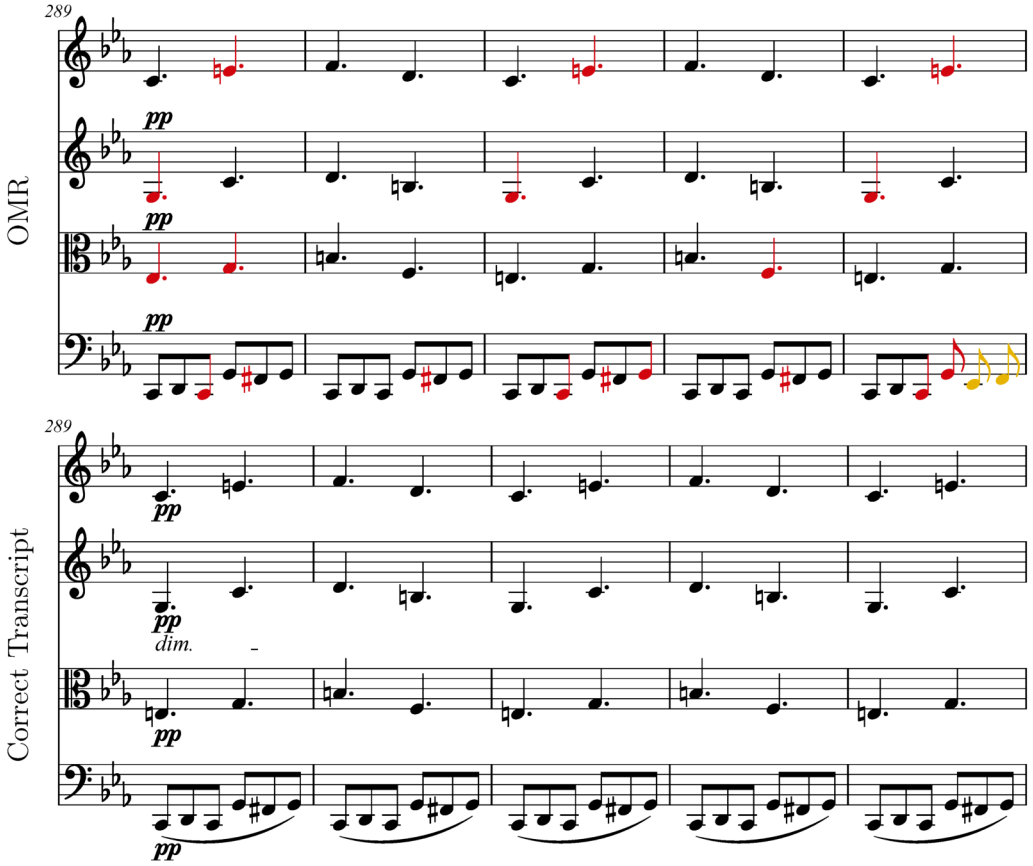
Replacing a dotted quarter rest with a quarter rest and an eighth rest would make no difference to the sound of the music when performed. It is possible that the transcriber did not see this kind of discrepancy as a priority, or that using quarter rests and eighth rests was considered more readable. The accent markings are a more complicated case, since the absence of an accent marking should, in principle, affect the realization of the piece in performance. One possibility is that the transcriber simply neglected to add those accent markings in. It is more likely, though, that they decided that explicitly adding an accent mark to every single quarter note in this section was redundant or not necessary, as a musician familiar with the genre would pick up on the pattern. This kind of omission of repeated symbols occurs with other types of glyphs too, most notably tuplet markings in sections with many consecutive tuplets that all use the same divisor. This speaks to the difficulty of finding sufficient amounts of clean training data when working with polyphonic music and machine learning, and also the difficulty

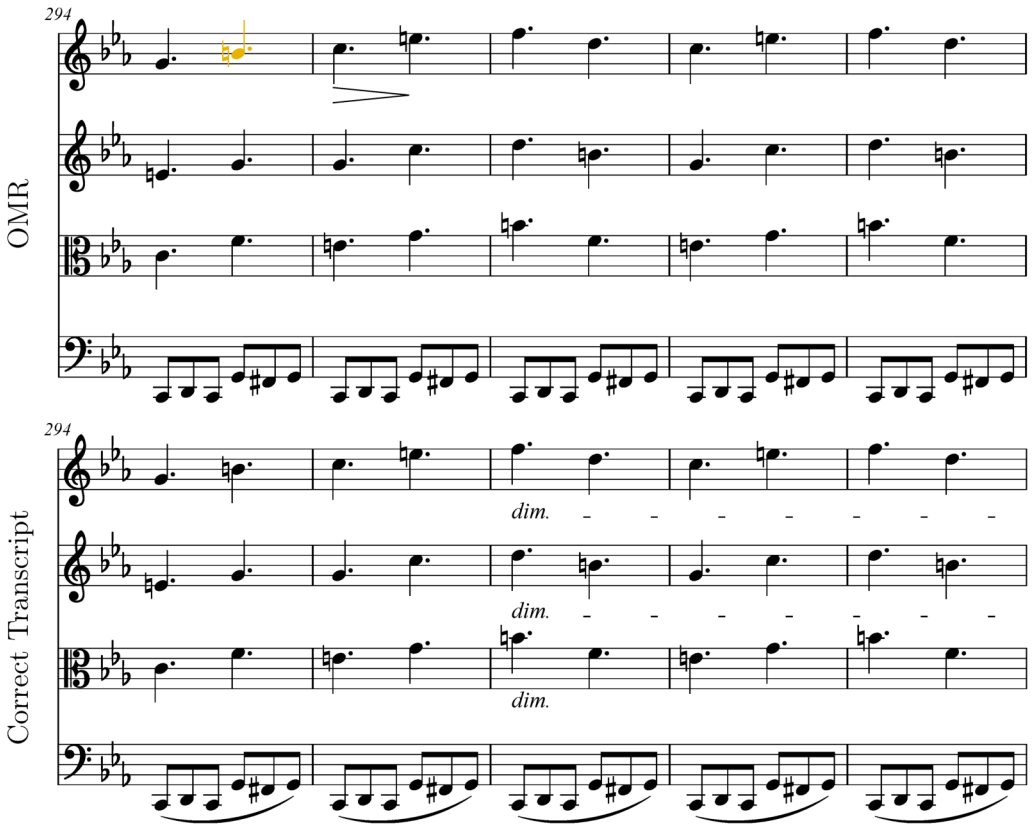
of evaluating OMR given the many extant standards for how to engrave Common Western Music Notation (CWMN).

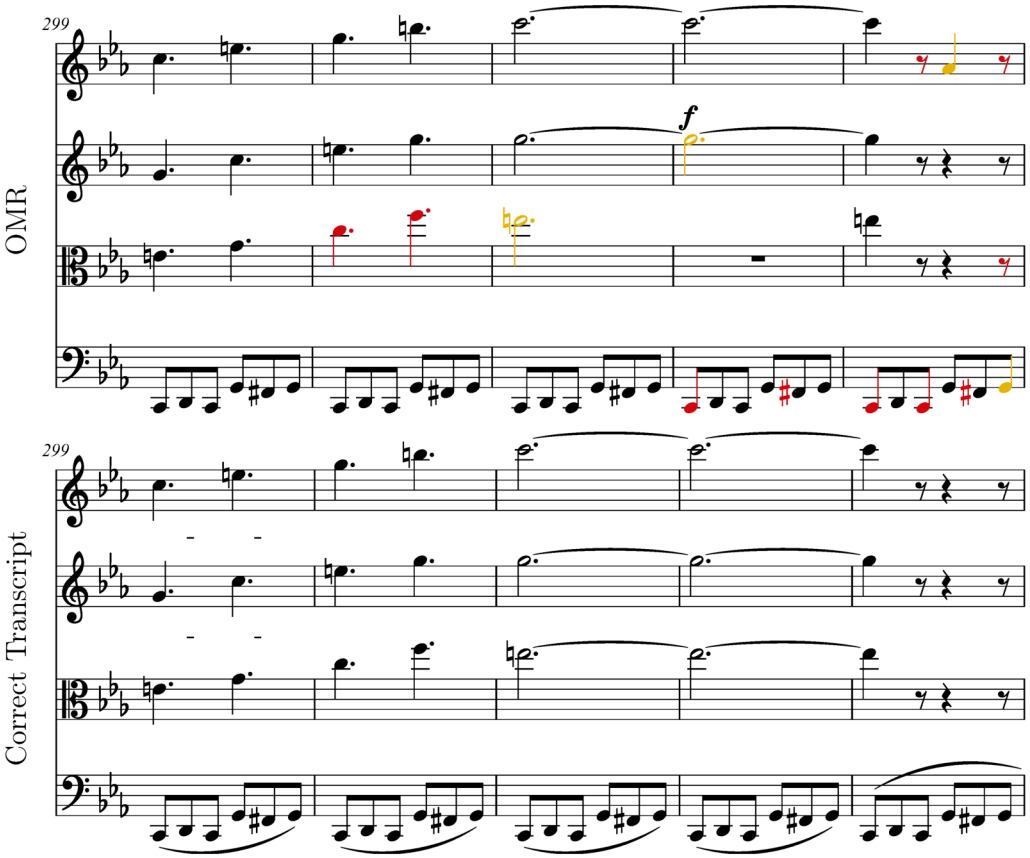
6.2.3 Franz Schubert - String Quartet No. 12 in C Minor

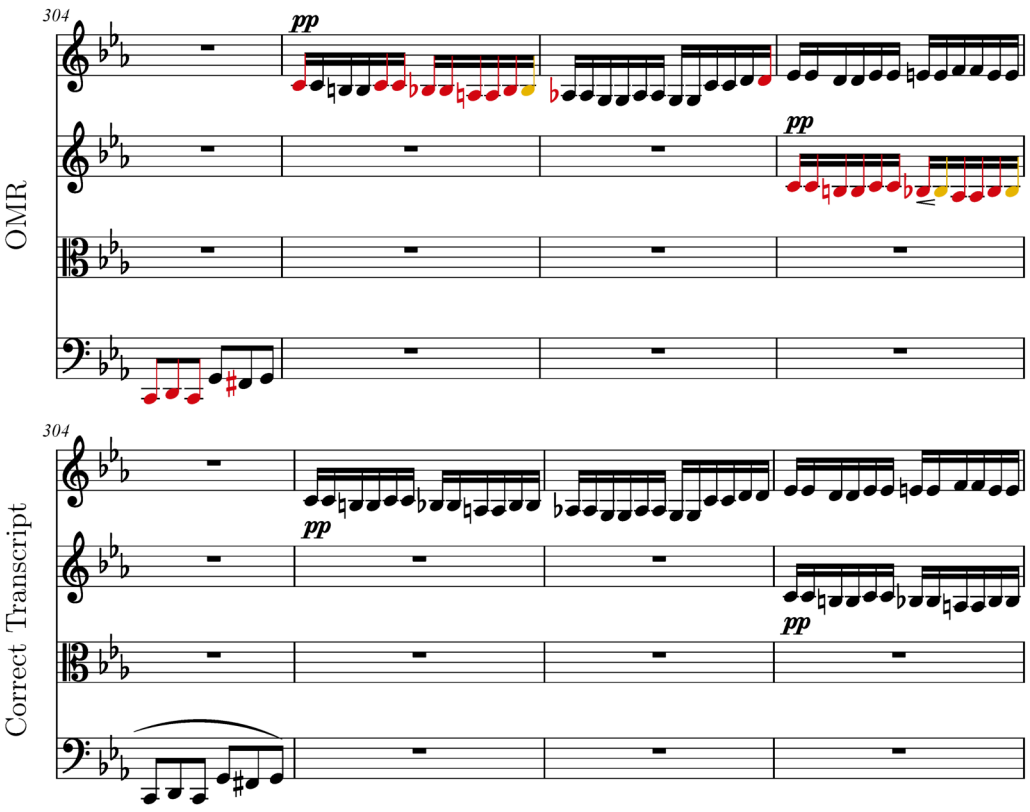
Franz Schubert's String Quartet No. 12 is a short piece (the first movement of an unfinished quartet) and is notationally about as complicated as the quartet from Fanny Hensel. The scanned score of it that I obtained is of a slightly worse quality and a low resolution. Figure 6.12 shows an extended excerpt from the OMR'ed version of the quartet with the results of the error detector highlighted, alongside the corresponding excerpts from the human-corrected transcript of the piece from the OSSQ.

The model performs well, making not a single false negative prediction within this excerpt. The most notable errors here occur when the original score notates a tremolo. PhotoScore does not often recognizes tremolo markings, instead marking them as single unbeamed notes, tending to fill the rest of the measure with rests to compensate. This happens often enough that the model is able to notice it nearly every time it happens and mark all failed tremolo detections.


OMR
289
pp
pp
pp
Correct Transcript
289
pp
pp
dim.
pp
pp


OMR
294
Correct Transcript
294
dim.
dim.
dim.


299
f
OMR
299
Correct Transcript


304
pp
pp
OMR
304
pp
pp
Correct Transcript

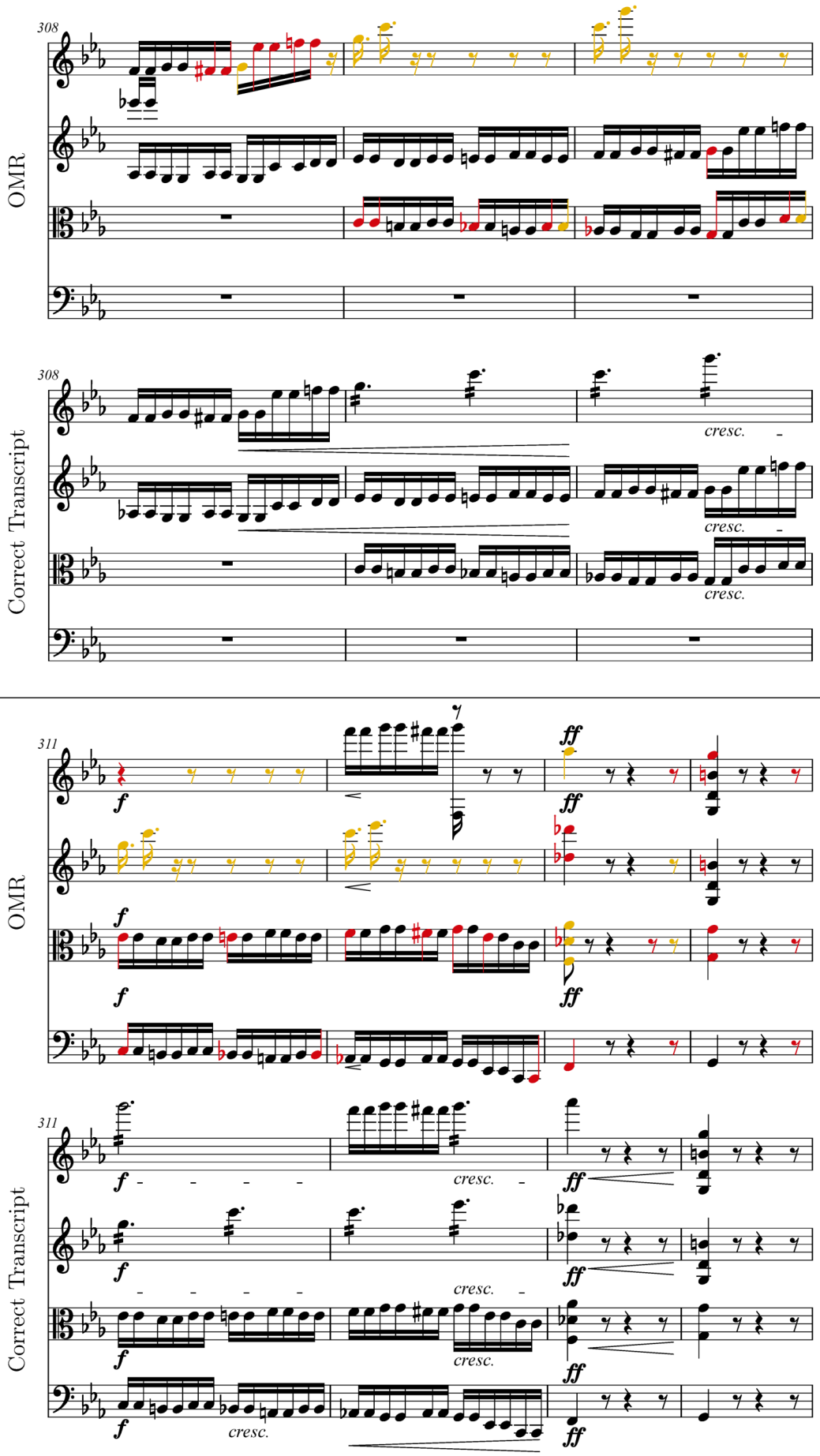


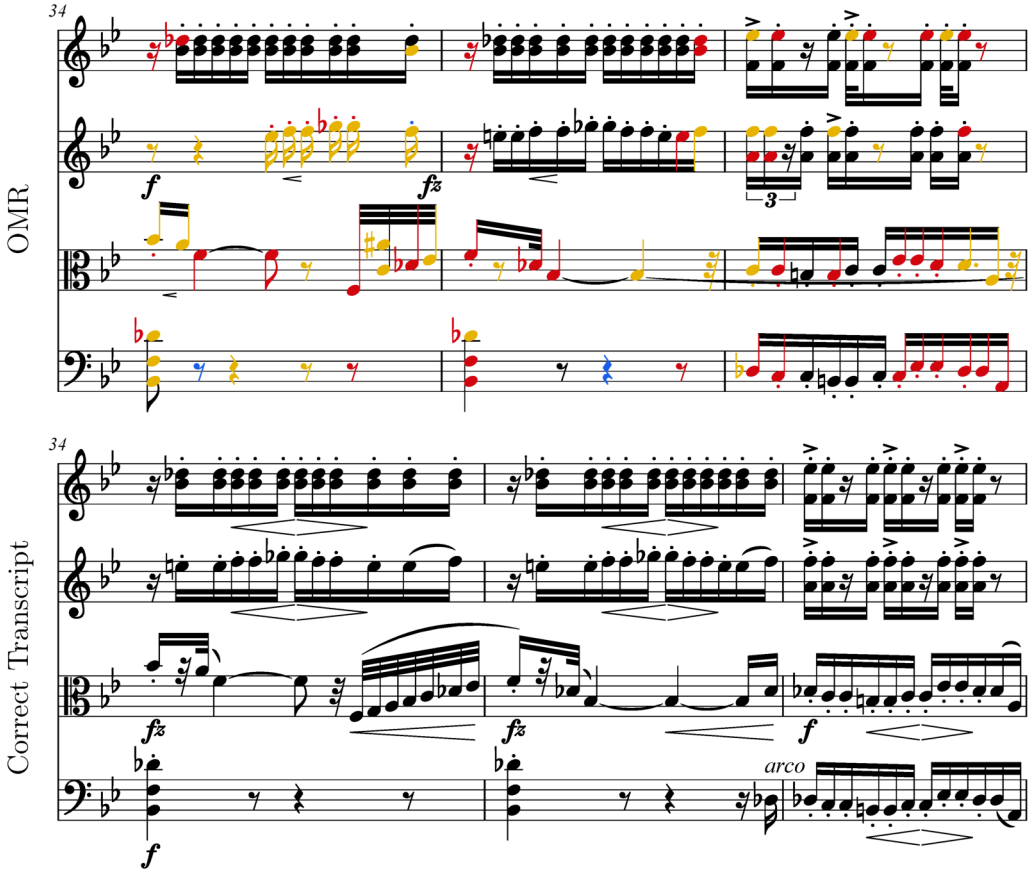
Figure 6.12: Franz Schubert's String Quartet No.12, D. 703, mm. 289–314. The OMR output from PhotoScore and the hand-corrected score are displayed side-by-side. Yellow symbols are correct detections, red symbols are false positives, and blue symbols are false negatives.

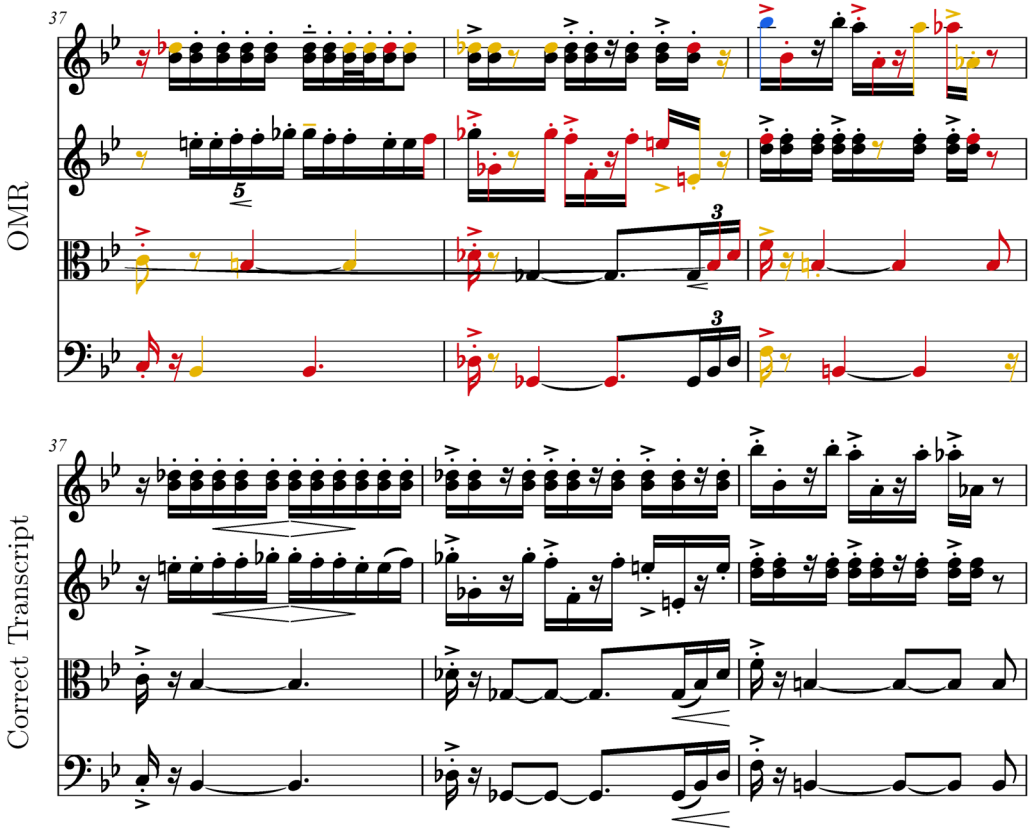
6.2.4 Edvard Grieg - String Quartet No. 1 in G Minor

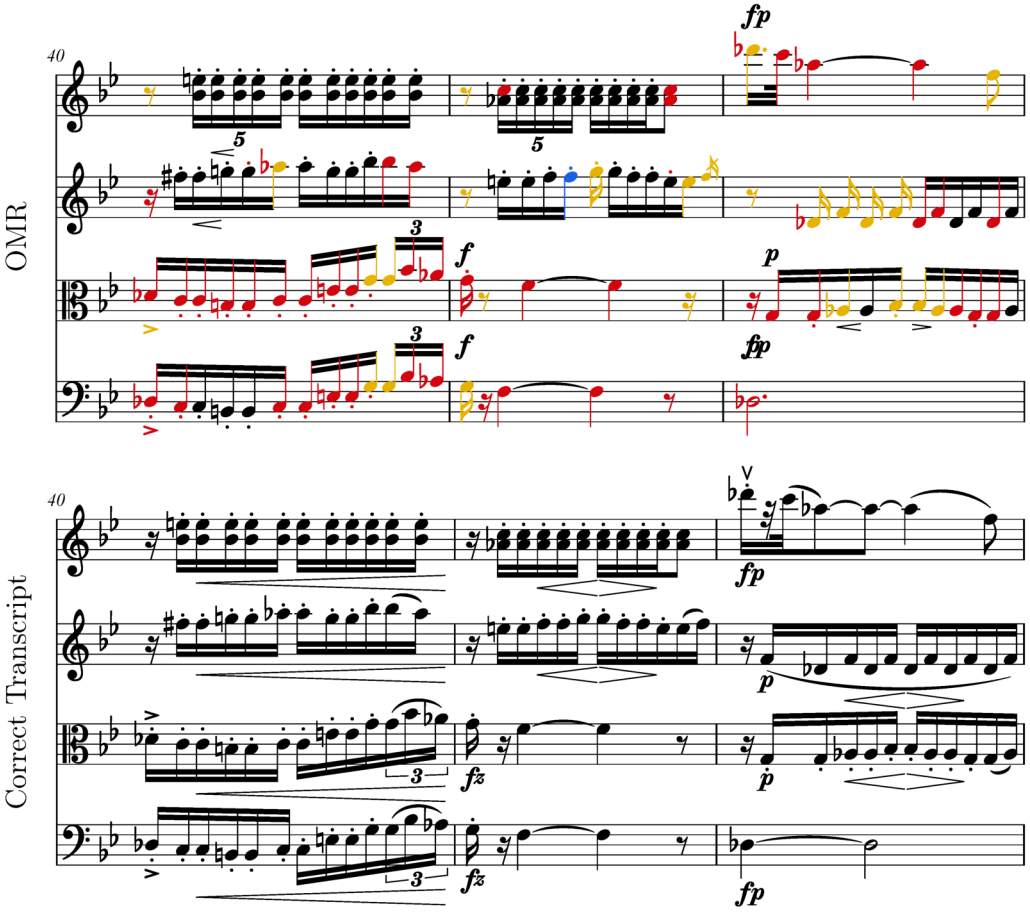
Edvard Grieg's String Quartet no. 1 presents a challenging scenario for OMR technology due to its intricate rhythmic patterns and dense texturing, including multiple double and triple stops played simultaneously by multiple instruments. The quality of the scanned score available is also significantly degraded, with staff lines faint and in some cases broken, and the engraving itself is also particularly dense, with a high number of measures notated on each staff line. These factors push PhotoScore to its limits in generating a semantically encoded file that is still readable by notation editors.

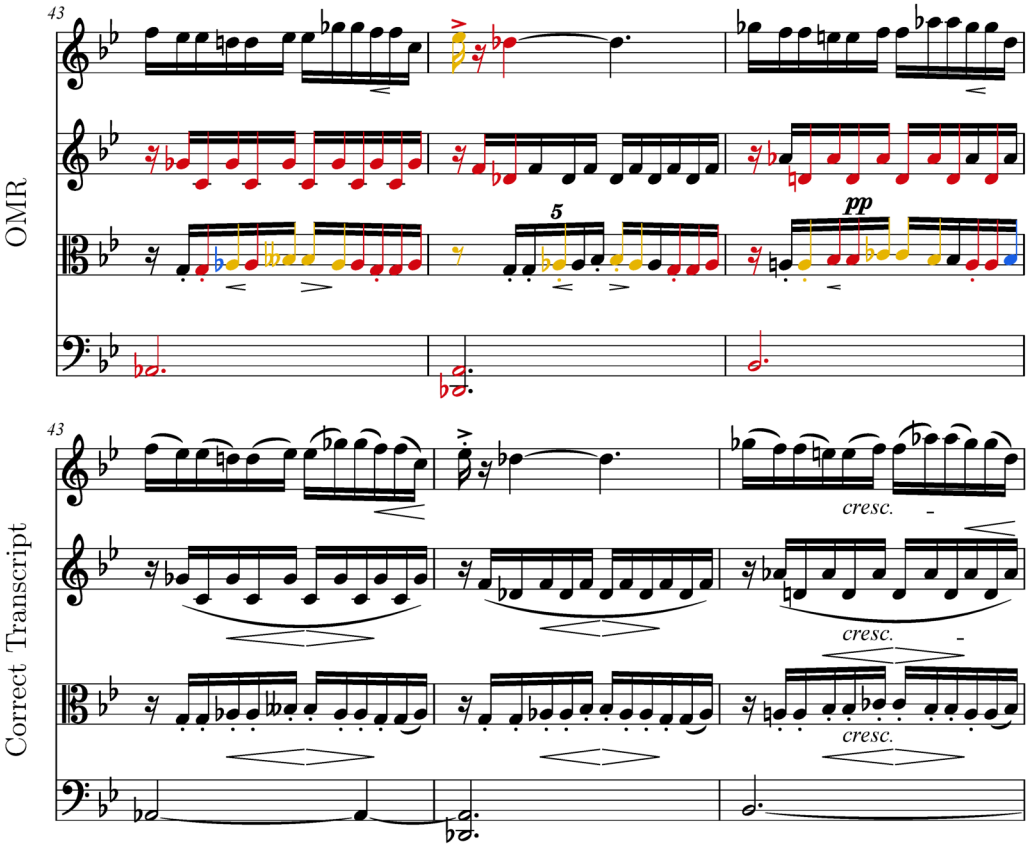
Figure 6.13 shows an extended excerpt from the OMR output with errors highlighted by the model, alongside corresponding excerpts from the human-corrected transcript of the piece from the OSSQ. Due to the large number of errors in the OMR output, much of the score is flagged as erroneous by the model. Many of these flagged errors would be easily identifiable by a trained transcriber without the model's help, calling into question the usefulness of these results. However, the model does successfully identify several sequences of sixteenth-note double stops as correct. It appears to rely on regularity and local repetition as indicators of correctness, suggesting that its output might offer at least some utility in excluding specific measures from further scrutiny by human transcribers.

One particular error occurs arises when Grieg writes beamed sixteenth notes with one note replaced by a rest, as shown in the first and second violin parts in the second system of Figure 6.13. PhotoScore's OMR often misinterprets this pattern, either by assuming a missing note or by treating the sequence as a 5-tuplet. These inaccurately generated 5-tuplets are consistently flagged as errors by the model, given the rarity of 5-tuplets in this musical genre and their frequent generation by OMR algorithms. Although the method used for visual representation in the figures cannot color tuplet markers, making them appear black, manual inspection of the model's raw output confirms that these are correctly marked as erroneous. In contrast, the notes within these false tuplets are not flagged as errors, as the agnostic encoding scheme treats them as normal sixteenth notes. Additionally, PhotoScore often outputs consecutive unbeamed sixteenth notes when it can determine note head positions and durations but fails to predict a valid beaming structure. Due to the frequency of this phenomenon in OMR outputs, the model is able to identify these errors as well.


OMR
34
Correct Transcript
34
arco


OMR
37
Correct Transcript
37


OMR
Correct Transcript
40
fp
f
p
fz


OMR
Correct Transcript
43
pp
cresc.

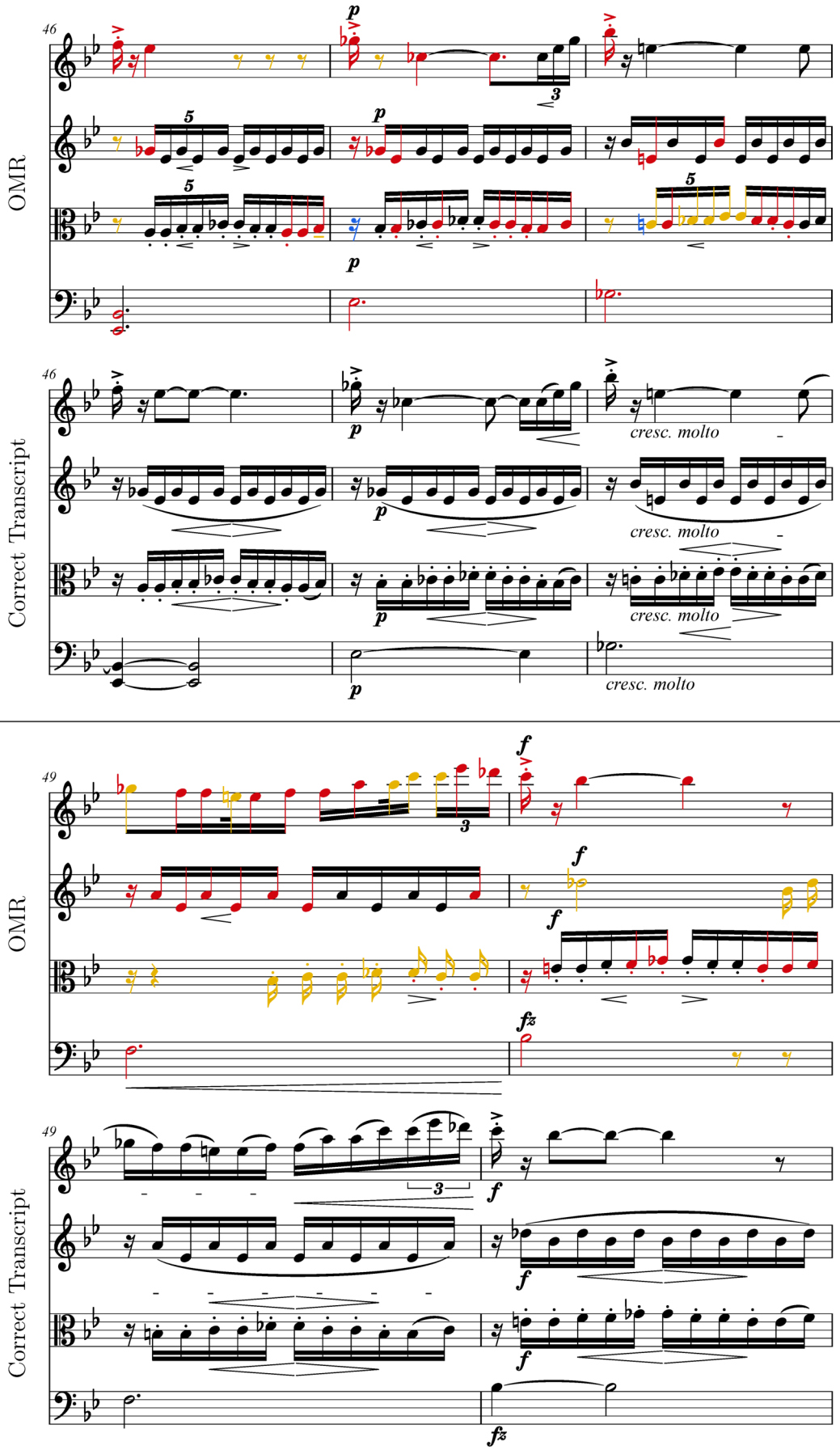


Figure 6.13: Edvard Grieg's String Quartet no 1, Op. 27, Mvt. 2, mm. 34–50. The OMR output from PhotoScore and the hand-corrected score are displayed side-by-side. Yellow symbols are correct detections, red symbols are false positives, and blue symbols are false negatives.

6.2.5 Scores Without Errors

As a sanity check, I also run some correct, error-less pieces through the model with the same settings. Ideally, the model should find no errors in pieces that are already human-corrected, but in practice, since it is tuned to minimize false negatives and tolerate false positives, it will still find some. This exercise also gives some insight into what kinds of musical phenomena the model thinks are most likely to be errors. Figure 6.14 shows the model's predictions when given a four-part Bach chorale harmonization and Figure 6.15 shows the model's predictions on the corrected version of a Mendelssohn string quartet. Symbols that the model has predicted as erroneous are colored red, as in these examples they are all technically false positive identifications.

The model predicts significantly fewer errors on these scores than it does on scores output from PhotoScore's OMR process, which is a promising result. From these and other examples, it appears that the model is most sensitive to the presence of possible errors in particular locations: at the beginning and ends of measures, at notes with accidentals (especially in dense passages), and in sequences of unusual durations or containing complex rhythms. Sequences of repeated notes or repeated short figures in stepwise motion are the least likely to be marked as errors.

6.3 Discussion

In this section I interpret the results presented in this chapter, highlighting noteworthy findings from these experiments. I then discuss the implications of these findings on the possible use of this error detector in large-scale music digitization efforts.

6.3.1 Effects of Data Augmentation

The results show that incorporating synthetic errors and natural errors is always beneficial to the model's performance over using just one or the other. However, the specific type of error augmentation employed appears to have little effect on the model's performance. The model trained with the simple error generation scheme (In the "Simple Data Aug" trial), which does not attempt to emulate the statistical distribution of real OMR errors, performs only marginally worse than the one trained with the full data augmentation pipeline. Further, the heuristics designed to cluster errors in a way that mimics real OMR data do not appear to affect model performance at all.

These results are surprising given the importance of carefully tailored error generation in

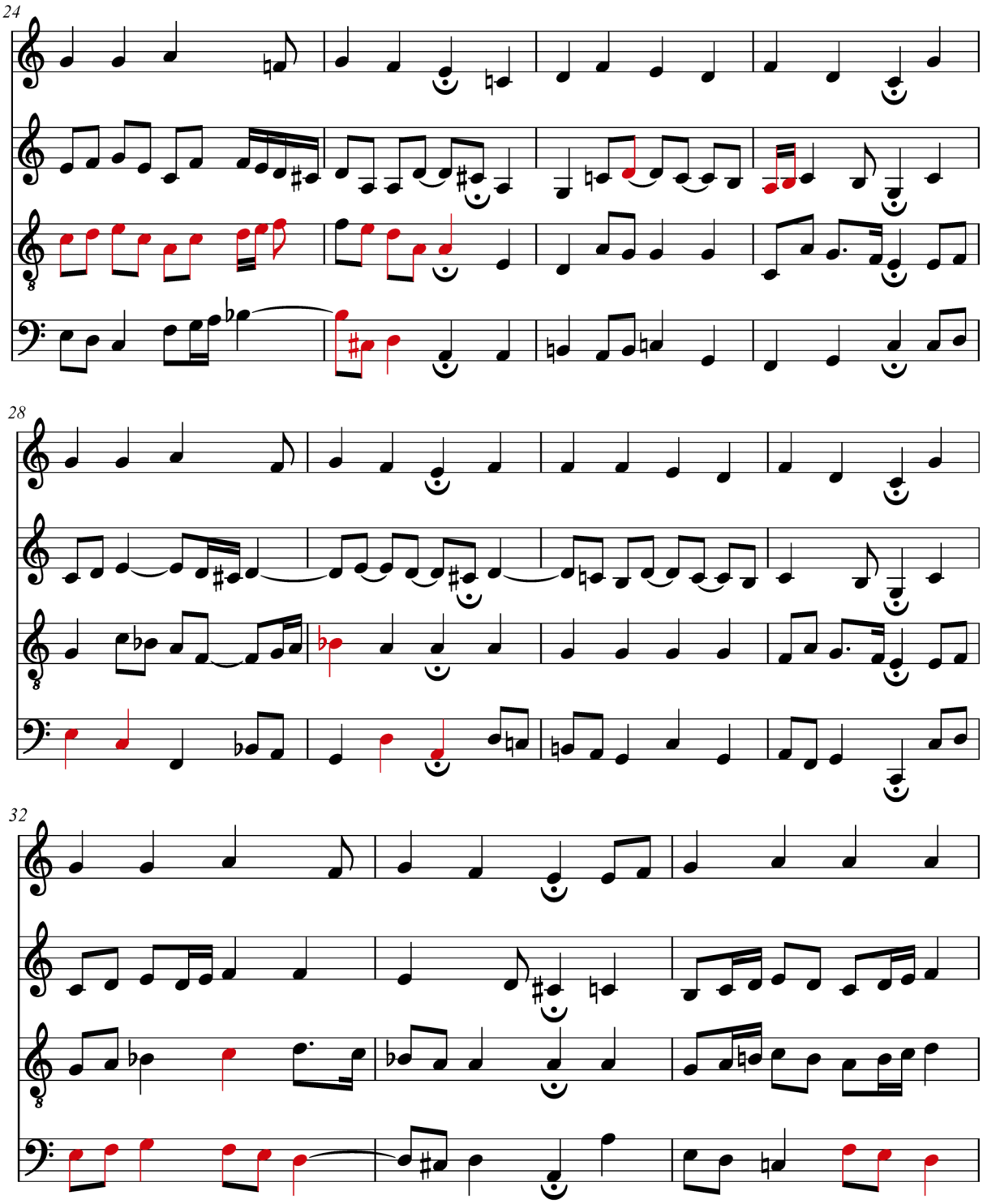


Figure 6.14: Johann Sebastian Bach, Chorale BWV 328, mm. 24–35, as run through the error detection model. This is a score without errors; there is no corresponding ground truth with OMR errors. The red-colored notes are where the error detection model has predicted "errors" might lie.

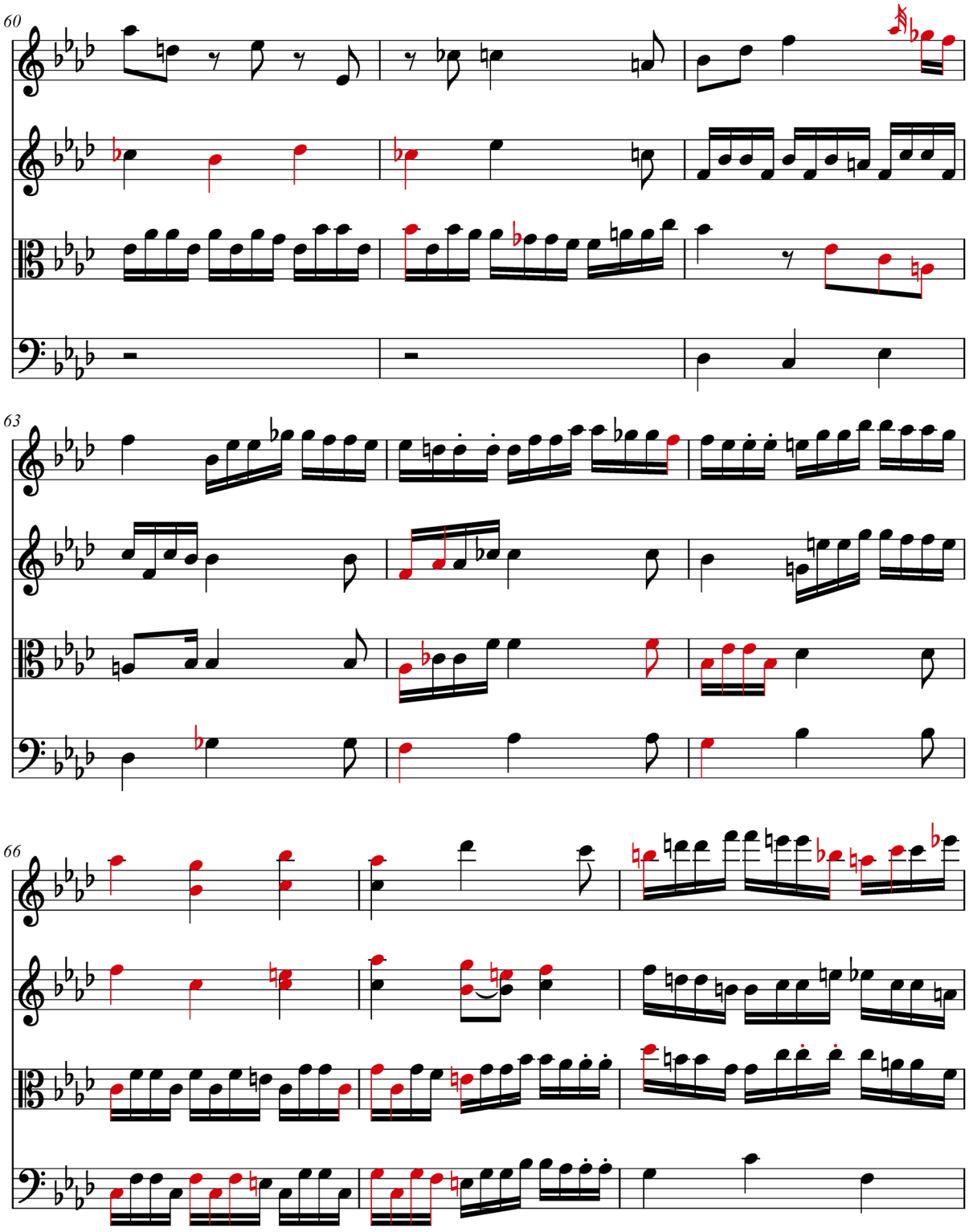


Figure 6.15: Felix Mendelssohn's String Quartet No. 5 Op. 44-iii, Mvt. 3, mm. 60–69. This is a score without errors; there is no corresponding ground truth with OMR errors. The red-colored notes are where the error detection model has predicted "errors" might lie.

related tasks. The fields of error correction and detection in natural language widely use data augmentation techniques that are finely tuned to generate synthetic errors that share the characteristics of natural errors, and these methods have been shown to increase the performance of error detectors when properly applied (See Section 2.2.1). It is possible, then, that the data augmentation procedure I have implemented simply does not generate errors that are similar enough to natural OMR errors for the model to benefit from them. On the other hand, Transformer decoder-based models like the Bidirectional Encoder Representations from Transformers architecture (BERT) and its many descendants (Devlin et al. 2019) have used randomly assigned errors to pre-train large models operating on natural language. However, BERT and its related models are intended for generative use, and not for error analysis. It is possible that more complicated data augmentation is simply unnecessary for the kinds of tasks these models are made for.

6.3.2 Sequence Length and Long-Term Dependencies

The other architectures tested in these experiments performed poorly compared to the LSTUT, with the exception of the LSTM, which had comparable performance. This suggests that, for this particular task, the stacked LSTMs are the most important components in the architecture of the LSTUT, rather than the Universal Transformer that lies between them. This is unexpected, especially considering the proven effectiveness of Transformer mechanisms in other music information retrieval tasks involving symbolic music. The vanilla Transformer, in particular, performs no better than chance, and both it and the Universal Transformer perform no better than the simple KNN model. Additionally, changing the input sequence length did not significantly impact model performance. These observations contradict my initial hypotheses that information on long-term repetition and structural patterns in Western classical music would aid the task of error detection, and that Transformers would excel in capturing this information. Several potential explanations exist for these outcomes:

- **The LSTUT architecture is not capable of learning long-term dependencies.** The LSTUT has been empirically shown to be able to reproduce medium- and long-term structures in solo piano scores (Berardinis et al. 2020). However, the model used in this research was much smaller (On the order of 1.4 million trainable parameters total). Additionally, the tasks were not the same as the tasks evaluated here; the LSTUT was developed with music generation in mind, and not analysis of existing symbolic music. It

is possible that the precise formulation of the LSTUT I have used, with a different set of hyperparameters, is not suited for the task I have given it.

- **The size of the dataset is insufficient for training a Transformer.** Transformers are well-known within the field of machine-learning research to be "data-hungry." More formally, this means that they lack the same inductive biases as convolutional and recurrent neural networks, and naturally place less assumptions on the structure of their input data. This allows them to model complicated phenomena that include distant parts of their input influencing one another, but this generality also means that they tend to have high numbers of trainable parameters, and they tend to only outperform simpler networks when trained with sufficiently large datasets. My dataset for fine-tuning has 360,000 notes, while the large dataset used to generate synthetic data has just under 2.3 million notes.[5] For comparison, the Music Transformer architecture (Huang et al. 2019) used a training dataset of 954 performances containing about 5 million notes, while Chou et al. (2021) presented a method for pre-training Transformer models on a combination of datasets totalling 6 million notes. Notably, Transformers trained on datasets of these sizes have failed to show evidence of learning notions of large-scale repetitive structure in music; see Dai et al. (2019), and the discussion of their work in Section 4.4.

- **The task of detecting the types of errors made by OMR processes does not benefit from the information given by longer sequence lengths.** It is also possible that the LSTUT architecture is capable of learning from long-term dependencies in musical data even with the relatively small size of the datasets used, but that the task itself did not necessitate it. In general, there is no reason to expect that a model should learn a representation of its input domain more complicated than it needs to be to accomplish the task it is trained for. If long-term repetition is simply *not useful* for correcting OMR errors, then it will not learn to look for long-term repetition. This would explain the negligible difference in performance between the LSTM and LSTUT architectures. Additionally, deGroot-Maggetti et al. (2020) showed that the presence of OMR errors in a score hinders the performance of pattern discovery algorithms. A score with a sufficient number OMR

5. These note counts are estimated from total token numbers of 875,043 and 5,613,095 for the small and large datasets respectively, and based on the statistic that about 41% of the agnostic tokens in each string quartet represent note heads. I use notes here to compare dataset size instead of tokens because different methods of tokenizing music result in vastly different numbers of tokens per note, and because note counts are the most commonly reported statistic on music dataset size.

errors may be degraded to the point of making an algorithmic analysis of its repeated sections impossible, or at least significantly less useful to the goal of error detection.

6.3.3 Usefulness to Human Correctors

Table 6.7 demonstrates that the model has the potential to reduce the time required for correcting OMR errors by an average of 20–30%, depending on whether the conservative or optimistic estimate of p is used. This potential for time reduction, however, varies significantly across different musical scores. Some scores show potential time savings of over 50%, while others are less than 10%. These results indicate that the error detector can substantially decrease correction times, but only in scores with few OMR errors and in music that is predictable enough for the error detector to accurately identify errors.

The assumptions underlying these estimates merit further consideration. These estimates disregard how different types of OMR errors require varying amounts of time to identify and correct them (as discussed in Section 3.2.2). In particular, these estimates do not differentiate between different distributions of an equivalent number of errors. For example, twenty consecutive errors on a single staff line are quicker to identify and correct compared to twenty isolated errors scattered across different parts and measures. Similarly, if the error detector marks a complex, non-contiguous subset of the score as erroneous, a human reviewer may take longer to examine the highlighted tokens compared to a situation where the marked errors are contiguous. This is evident in Figure 6.13, where the model's predictions can be erratic and scattered, particularly in dense scores with many errors, requiring a reviewer to check content from every measure. This can occur even when the model technically achieves a high precision score.

Equation 3.6, employed to estimate the time savings offered by the error detector, assumes that the detector achieves near-perfect recall. This implies that users can trust the model to not overlook any errors, thereby obviating the need to scrutinize unmarked sections of the score. However, achieving this level of recall necessitates setting the detection threshold so low that almost the entire composition might be identified as erroneous. As indicated in Table 6.4, to reach a recall rate of 0.99, most models must classify at least 86% of the score as erroneous. In this context, I have presumed that a recall rate of 0.9 is sufficiently high for the error detector's results to remain valuable. With this recall rate, one in every ten errors may go undetected, as observed in the illustrated examples, Figures 6.11 and 6.13, which also operate at a recall rate of 0.9. However, a majority of these undetected errors (false negatives) tend to be located adjacent

to material the model has marked as erroneous, suggesting that the model often successfully identifies the general area of an error, even if it cannot precisely locate the erroneous token.

This insight leads to the conjecture that achieving a recall rate near 1.0 might not be necessary for the error detector's output to significantly expedite the correction process. In scenarios where a measure contains multiple true positives alongside a solitary false negative, the error detector's indications are likely sufficient to draw the corrector's attention to the relevant measure, enabling them to identify the overlooked error. This is particularly plausible since errors that are completely missed by the detector (false negatives within long sequences of true negatives) are relatively uncommon. To further refine these estimates of time savings and to validate whether a recall rate of 0.9 is adequate, conducting user trials would be necessary. Such trials would provide empirical data on how users interact with notation software during the OMR correction process, including the time taken to identify and correct various types of errors. This potential area of research is discussed in Section 7.3 as a direction for future studies.

The final roadblock to the usefulness of the detector is that even when its predictions are correct, the model's output is not intuitive to interpret. See Figures 6.9 and 6.10 and the accompanying discussion for examples of how the detector's predictions can be unituitive to read. This arises from the necessity of transforming polyphonic, multi-staff scores into one-dimensional sequences. Such a transformation inevitably disrupts the two-dimensional spatial relationships inherent in musical scores. Some elements that are musically close together, like simultaneous events across different staves, become separated in the encoded agnostic sequence. Conversely, events that are adjacent in the agnostic sequence might be distant in the actual score, such as elements at the end of one measure and the beginning of the next measure on a different staff. One potential approach to address this issue would involve encoding each instrument's staff individually before concatenating these sequences. This method would prevent misinterpretations of deleted elements at the beginning of one measure being identified as errors at the end of the preceding measure. However, this solution also has the drawback of eliminating the model's ability to learn from vertical harmonies and polyphonic textures, fundamental aspects of Western classical music.

Ultimately, this challenge appears to be an inherent limitation of the methodology employed. The complexity and hierarchical nature of musical scores make it challenging to represent them in a format that is both conducive to machine learning and intuitively interpretable by an end-user. Exploring alternative methods for encoding musical scores, which could potentially

address these issues, is an area for future research. Options for such encoding methods and their implications for the model's effectiveness and interpretability are further discussed in Section 7.3.

Chapter 7

Conclusions and Future Work

This chapter concludes the dissertation. Section 7.1 summarizes the research that has been presented, discusses principal results and contributions, and evaluates whether the error detector meets the goals I set out for it. Section 7.2 describes the publicly available repository containing resources and datasets necessary to replicate this research. Section 7.3 discusses possible extensions to this research, including adjustments to the methodology, different methods of representing musical data in the model, and further trials using human correctors to evaluate the efficacy of the system.

7.1 Summary of this Dissertation

The research detailed in this dissertation aimed to expedite the time-intensive process of converting scanned musical scores into machine-readable symbolic music files. The underlying hypothesis was that most errors generated by Optical Music Recognition (OMR) processes are noticeably un-musical, and so a machine-learning model could identify and highlight these errors without the need to cross-reference the original score. It was anticipated that if the detector could reliably identify errors, a human reviewer could confidently ignore unmarked regions, assuming they are error-free, thus reducing the overall time required for OMR output correction. Additionally, I hypothesized that synthetic errors, designed to mimic the characteristics of natural OMR errors, could be used to effectively train the model. This approach was proposed as a solution to the lack of extensive datasets pairing original OMR outputs with their corrected versions, which would otherwise be costly and time-consuming to compile. Given that music typically displays a high amount of repetition, I further hypothesized that a machine-learning

architecture capable of analyzing long-term dependencies could use those repetitive qualities to identify errors by noting irregularities in the repetition.

In Chapter 2, the existing work on error and outlier detection in sequences containing real-valued data and natural language was surveyed. This domain is similar to symbolic music, providing a foundational context for the design of the error detector. The chapter also described sequence alignment, focusing on the Affine Needleman-Wunsch (ANW) alignment and the algorithms for its computation. Additionally, the chapter outlined a range of works and research directions that specifically address error analysis in music.

Chapter 3 offered an in-depth background on OMR, emphasizing the types of phenomena in score images that lead to errors in OMR processes and the propagation of these errors into encoded outputs. The commercial OMR application PhotoScore was examined comprehensively to contextualize the traditional OMR workflow from a user perspective. I developed a method based on the Keystroke-Level Model (KLM) of user interaction to estimate the potential time savings offered by the error detector.

In Chapter 4, I described binary classification tasks, the process of thresholding, and the challenges inherent in developing interpretable and effective error metrics for classification tasks like error detection. This was followed by a comprehensive overview of machine learning, beginning with fundamental principles such as backpropagation and Stochiastic Gradient Descent (SGD), and culminating in a detailed examination of recent variants of the Transformer architecture.

In Chapter 5, I developed a methodology to define the task of error detection in a manner applicable to any score written in Common Western Music Notation (CWMN). This process began with the creation of a method for encoding polyphonic, multi-staff scores into one-dimensional sequences of agnostic tokens. It also involved devising a procedure for converting semantically encoded files into these agnostic sequences. Subsequently, a method was established for deriving a "difference" between two agnostically encoded sequences using the ANW alignment. Using this method, I generated binary labels that identify individual tokens in OMR output as either correct or erroneous. I then defined the dataset of string quartets utilized for model evaluation, along with various data augmentation procedures to enhance the effective size of the datasets by introducing synthetic errors into a corpus of correct string quartets.

Chapter 6 involved the design and execution of a comprehensive set of trials to assess variations of the proposed architecture and compare them to simpler baselines. These trials evaluated the method's performance along various dimensions, including different machine-learning

architectures, data augmentation methods, dataset configurations, and input sequence lengths. The primary machine-learning architecture I used was the Long Short-Term Universal Transformer (LSTUT), which was compared against the vanilla Transformer, the Universal Transformer, the Long Short-Term Memory (LSTM) network, and a method based on K-Nearest Neighbors (KNN). This exhaustive evaluation aimed to rigorously test the effectiveness and versatility of the developed error detection method in the context of OMR.

Results indicated that the LSTUT model, when pre-trained on synthetic errors and subsequently fine-tuned on natural errors, achieved the best overall performance for the error detector. Other training variables, such as the type of data augmentation or the length of input sequences, had minimal impact on the model's effectiveness.

Employing the KLM-based method for potential time savings estimation, I calculated that the error detector's annotations could potentially halve the time required by a human corrector to rectify OMR output. However, this estimate varied significantly based on the specific musical piece. The utility of the error detector was greatest when operating on pieces with relatively few OMR errors in the output. In such cases, the corrector must spend most of their time identifying errors, a task which is significantly expedited by the error detector. The detector's proficiency in identifying all errors in a score was higher in instances with fewer errors. In contrast, the estimated time savings were substantially lower for scores with a high frequency of OMR errors. This reduction in time saved can be attributed partly to the lower proportion of time spent by correctors in searching for errors and partly to the diminished performance of the model under these conditions.

These experimental outcomes corroborated the initial hypothesis that an error detector could feasibly reduce the time required for correcting OMR outputs. As OMR technologies advance and generate fewer inaccuracies, the potential utility of the error detector, in terms of time efficiency, will grow even further. The approach of using synthetic data in the training process was somewhat validated, as the models that were trained using both synthetic and natural OMR errors exhibited superior performance compared to those trained exclusively on natural errors. However, models trained only on synthetic errors showed poor performance, indicating the necessity of a moderately sized dataset of natural OMR errors for the effectiveness of this approach. Contrary to expectations, the importance of information on long-term dependencies in identifying errors was not substantiated. Variations in the length of input sequences had negligible impact on the model's performance, suggesting that the detection of errors in music

may not rely as heavily on understanding long-term dependencies as was initially hypothesized.

7.1.1 Contributions

The novel contributions of this research have been:

- **A method of detecting errors in symbolic music suitable for OMR-related applications.** The error detection models developed in this research are publicly available via a command-line interface for inference on new symbolic music files (see Section 7.2).

- **A general and extensible definition of errors in symbolic music.** The notion of error introduced in Section 5.2.4, defined by taking the ANW alignment between two musical sequences, is to my knowledge the first such application of sequence alignment in music that is capable of handling all possible elements that can appear in CWMN. This definition is simple to implement and broad enough to work on any music for which representation as an agnostic sequence is possible.

- **An evaluation of data-augmentation techniques for the training of OMR-related tasks.** I implemented three types of data augmentation: one that created random synthetic errors, one that matched the distribution of error types from the test set of natural OMR errors, and one that included heuristics to match how errors tend to cluster together in the test data. These were all evaluated by pre-training models on synthetic errors before fine-tuning them on natural errors. Scarcity of data is a significant issue in nearly all machine learning fields that operate on symbolic music. Data augmentation techniques for symbolic music are not as mature and well-studied as they are in Natural Language Processing (NLP).

- **Evaluation of a range of machine-learning architectures capable of learning long-term dependencies on musical data.** The error detection task was evaluated on the LSTM network, the vanilla Transformer, the Universal Transformer (UT), the LSTUT, and a baseline using KNN. Despite my use of Transformer networks, there was little evidence that the error detector used information on long-term musical repetition to make judgments. This negative finding is interesting and hopefully informative to other researchers trying to design machine-learning algorithms to retrieve information from symbolic music.

7.1.2 Design Goals Met

In this section, I revisit the design goals outlined in Section 5.1 for the error detector and assess the extent to which each goal has been successfully achieved.

- **Genre and Instrumentation Agnosticism: Achieved.** The agnostic encoding method, as outlined in Section 5.2.3, can represent any music written in CWMN, rendering the error detector genre and instrumentation agnostic. This universality allows the detector to potentially operate on a wide range of musical compositions, although its performance would likely vary across different genres, particularly those that diverge from traditional methods of music typesetting and formatting.

- **Prioritizing High Recall Rate: Partially Achieved.** The error detector, at recall rates of 0.99, has such low precision that it would not speed up the process of error correction. At slightly lower recall rates of 0.9, its precision is high enough to effectively aid human correctors in identifying errors, if we assume that this would not, in practice, result in errors going unnoticed; see Section 6.3.3 for an argument that 0.9 recall may be sufficient. This achievement aligns with the objective of reducing the time and effort required in the error correction process.

- **Granular Error Localization: Not Achieved.** The detector's theoretical capability to pinpoint exact error locations was not fully realized in practice. Instead, it tended to identify the approximate areas of errors, often marking more tokens as erroneous than necessary. The impact of this limitation on the overall utility of the detector in the correction process is uncertain. Specifically, the advantage of more precise error localization in enhancing the efficiency of the correction process remains unclear.

- **Feasibility with Limited OMR Error Data: Partially Achieved.** Data augmentation strategies were proven to enhance the performance of the error detector, particularly when a model trained on synthetic errors underwent fine-tuning with natural errors. Surprisingly, the specific approach to data augmentation used did not significantly impact model performance, indicating that a large corpus of OMR errors is not mandatory for training an effective synthetic error generator. However, the necessity of incorporating natural OMR errors for training suggests that a baseline dataset of such errors is essential for optimal model function. Data augmentation, while beneficial, cannot completely

substitute the need for natural error data.

- **Adaptability to Small Music Corpora: Not Achieved.** The error detector's effectiveness in adapting to genres with limited digital corpora was not conclusively demonstrated. The necessity for a substantial dataset of natural OMR errors to train the model limits the application of this method to musical genres with a limited number of digitized works. While data augmentation techniques used in this research helped to some extent, they were insufficient to entirely mitigate the need for natural error data. To enhance the adaptability of the model to genres with smaller digital corpora, further research in data augmentation methods is necessary. Additionally, demonstrating the model's ability to generalize error detection across different genres, after being trained on one specific genre, could significantly enhance its utility in this area.

- **Ability to Process Long Inputs: Not Achieved.** Despite the model's technical capacity to take long input sequences as input, its actual utilization of long-term dependency information in decision-making was limited. This underutilization of the model's input processing capabilities is further elaborated in Section 6.3.2.

In summary, while the error detector exhibited some adaptability to the limitations of OMR error data and music corpora size, its effectiveness in leveraging long input sequences for error detection was not realized. The model's reliance on a foundational set of natural OMR errors for optimal training highlights a key area for future development, especially for musical genres with smaller digital repositories.

7.2 Source Code and Training Data

All of the code used to train models, perform parameter sweeps, evaluate models on test data, and generate engraved figures (including the examples in Section 6.2) is publicly available in a GitHub repository.[1] The code is written in Python 3.9.12, and requires several key libraries:

- PyTorch (Paszke et al. 2017): A widely used machine-learning framework that provides flexibility and efficiency in model development and training.

- Music21 (Cuthbert and Ariza 2010): A toolkit for parsing and managing semantic music files, crucial for handling various musical data formats.

1. `github.com/timothydereuse/transformer-omr-spellchecker`

- Python bindings for Verovio (Pugin et al. 2014): Used for generating engraved outputs with highlighted features, useful for visualization and analysis of the model's performance.

- Numba (Lam et al. 2015): A just-in-time compiler for Python, used for an efficient implementation of the ANW alignment algorithm.

The parts of the code that implement the semantic-to-agnostic conversion and the implementation of the ANW alignment are standalone modules that other researchers are free to use in other projects.

The datasets of natural OMR errors were assembled manually, by pairing up human-corrected scores with versions I ran through PhotoScore's OMR process, as described in Section 5.3. These were all taken from the OpenScore project (Gotham et al. 2023), which transcribed classical works from scans in the public domain and released the transcriptions into the public domain. These scores are supplemented with scores from the Mendelssohn String Quartets Database (deGroot-Maggetti et al. 2020), whose results are also freely distributable under the Creative Commons BY 4.0 License[2]. This entire portion of the dataset is available for download, including scanned scores in Portable Document Format (PDF) format, scores with OMR errors, and the original human transcriptions, via a link in the repository mentioned above. The dataset of string quartets not containing OMR errors, used to generate synthetic data, is a union of other datasets, described in detail in Section 5.3. One of these datasets was downloaded by myself and comprises scores freely available from the MuseScore score distribution platform. Most of these scores are in the public domain, but some are not assigned a license marking them as freely redistributable. For this dataset, I redistribute only those works that have been explicitly marked as being in the public domain. This dataset is also available via a link in the project's repository.

To use the code, the user must first run a script that converts all of the data to an agnostic format and creates labeled testing data by aligning the paired OMR / human-corrected quartets. Users can use their own datasets by replacing the MusicXML files of the datasets provided with the repository, while keeping the exact same directory structure. Then, a model can be defined by changing one of the parameter files to set hyperparameters, training times, and the parameters of the machine-learning architecture. Scripts for testing the file and using it to evaluate unseen symbolic music files are also provided. Details are included in a README file bundled with the

2. `creativecommons.org/licenses/by/4.0/`

code.

7.3 Future Work

Here I discuss possible avenues for evaluating the detector, improving its performance, and potentially using it for other applications. These possible future paths include using a hierarchical graph representation for music (Section 7.3.1), letting the detector represent errors more informatively (Section 7.3.2), methods for visualizing the predictions of the detector in a correction interface (Section 7.3.3), new methods for collecting or augmenting training data (Section 7.3.4), application of the methodology to correction Automatic Music Transcription (AMT) output (Section 7.3.5), and user studies (Section 7.3.6).

7.3.1 Using a Hierarchical Graph Representation for Music

A fundamental component of the error detection approach I propose involves converting musical scores into one-dimensional sequences of agnostic tokens, as detailed in Section 5.2. This transformation is crucial for several reasons: Representing scores as one-dimensional sequences enables the application of the ANW algorithm to accurately pinpoint error locations within the music. By treating sequences of agnostic tokens similarly to textual data, this format also allows for the direct application of techniques from the field of NLP to this task. Lastly, there is a more extensive range of established machine-learning architectures designed for one-dimensional sequence data compared to those that handle hierarchically structured data with complex shapes. This compatibility broadens the scope of potential machine-learning solutions that can be applied.

However, as discussed in Section 5.2.4, this process of converting scores into agnostic token sequences significantly influences the nature of the results. The act of "flattening" music's inherent hierarchical structure inevitably introduces certain "un-musical" effects. These effects stem from how a serialized representation must inevitably obscure the hierarchical and spatial relationships inherent in traditional musical notation. Navigating these side effects is a critical aspect of the design and implementation of the error detection method, ensuring that it remains musically informed and effective in identifying errors.

There are machine-learning models specifically designed for graph or tree-structured data. These models can naturally accommodate the hierarchical relationships intrinsic to musical sym-

bols, harnessing these connections within their algorithmic framework. The concept of designing machine-learning architectures to align with the inherent structure of specific data types is known as Geometric Machine Learning (Bronstein et al. 2021). Convolutional neural networks are an example of an architecture that operates on geometric principles, as the operation of convolution itself encodes information on how pixels of an image are adjacent to one another; pixel adjacency is necessary information for a network to be able to interpret images. In contrast, a fully connected feed-forward layer, when applied to image data, does not inherently provide the network with information into pixel adjacency. The field of symbolic music processing may stand to gain significantly from designing machine-learning models with these structural regularities in mind. By using models that inherently represent the structural and hierarchical nuances of musical scores, researchers and practitioners in music-related machine learning can achieve more powerful and contextually aware interpretations and analyses.

A new method of representing music would necessitate a new method of representing errors in music. There exists a field of research on graph alignment methods that, in an analogous way to how the ANW algorithm matches up regions between two related strings, find nodes and edges that correspond to one another in related graphs. The difference operation, in this case, entails identifying the necessary graph operations (such as adding or deleting nodes or edges) to transform one musical graph into another (Delugach and Moor 2005). This concept opens up the possibility of formulating a "difference operation" for music represented as graphs, suitable for use in a machine-learning model that operates on graph- or tree-structured data. With such a graph difference operation, the methodologies discussed in this dissertation could be adapted with minimal alterations to the context of geometric machine learning.

7.3.2 Methods for Representing Errors

While my current method was focused on a singular category of error prediction, it would be feasible to extend it to predict multiple types of errors; for instance, the model could be refined to differentiate between insertions, deletions, and replacements of tokens. Further, with some additional modifications, it could categorize various kinds of replacements, like discrepancies in pitch, duration, stem direction, or beaming. When it comes to detecting tokens deleted in the OMR process, the model might also estimate the number of tokens that should be inserted. Introducing these extra categories provide a more nuanced and interpretable visualization of results for the users, and would necessitate no significant architectural modifications to the

model. However, as noted in Section 5.2, preliminary experiments with an error predictor that differentiated errors into deletions, insertions, and replacements showed poor performance, especially in detecting deleted tokens. Given this, I prioritized achieving higher accuracy on a simpler task over lower performance on a more complex task. As discussed in Section 5.1, the practical utility of such detailed error categorization in a correction-aided context is unclear.

With the foundation established by this research, however, it could be worth investigating whether extending the method to incorporate more intricate error categories could provide tangible benefits to human correctors. Additionally, the musically counterintuitive nature of the agnostic encoding method might be offset by a more sophisticated approach to presenting results to end users that includes more details on the nature of the errors detected.

7.3.3 Methods for Visualizing Errors in a Notation Editor

The current method of outputting results in a separate MusicXML file and rendering them using Verovio is not ideal from a user standpoint. In a real correction scenario, this would still require the user to look back and forth between the results of the error detector and the notation editor used to correct the score. Integrating the error detection results directly into a score correction and editing interface would substantially enhance its practical utility. This approach, akin to the integration seen in PhotoScore's OMR correction interface (discussed in Section 3.3.2), would provide users with an intuitive method of interacting with the error detector's predictions.

I have explored the possibility of creating a user extension to the MuseScore notation program that allows for interactivity with the results of the error detector. Such an extension could employ dynamic coloring or highlighting of the score via MuseScore's extension Application Programming Interface (API). Key features could include allowing users to modify the threshold for error detection to suit specific pieces, and providing the option to reprocess the edited piece through the error detection model and update the score's coloring based on the latest edits. MuseScore extensions are written in JavaScript, which supports libraries and frameworks capable of running machine-learning models in inference mode. A review of MuseScore's extension development documentation suggests that arbitrary coloring of score elements is feasible through the API, so creating this extension is theoretically viable. However, my limited experience with JavaScript application development and unfamiliarity with the MuseScore developer API meant that I could not meaningfully assess how difficult it would be to create such an extension, or what kind of technical issues would be encountered in its development. Therefore, while this

integration concept shows significant promise for enhancing the usability and functionality of the error detection method, I recognize it as an area for future development.

7.3.4 Training Data and Data Augmentation

The data augmentation method I designed was intentionally crafted to be both simple and efficient, enabling on-the-fly error generation during training without creating a computational bottleneck. Given the relatively modest size of the datasets utilized, my hypothesis was that offering the model a virtually unlimited array of variations on the string quartets would be advantageous. Additionally, I aimed to avoid allocating a significant portion of the OMR training data to develop the synthetic error generator, anticipating that a more complex model-based approach might overfit to the limited data available.

An alternative method for enhancing the quality of the generated data could involve using a sequence-to-sequence translation model. This model, trained on OMR outputs, would effectively "translate" corrected scores into versions with OMR-like errors. Implementing such a model would likely necessitate pre-generating a large amount of augmented material in a separate dataset, rather than creating errors during training. The main difficulty with this approach would be training an effective translation model using a very small amount of data. Most models in the fields of Grammar Error Correction (GEC) and Grammar Error Detection (GED), which use similar back-translation strategies, operate on natural language datasets significantly larger than what is typically available for symbolic music, and so can use sophisticated models with large numbers of parameters without risk of overfitting. Hence, while a sequence-to-sequence model might theoretically produce more realistic OMR-like errors, the feasibility of this approach is uncertain under present data availability constraints.

A more promising direction would be to find a way to significantly speed up the process of automating generation of natural OMR errors. The ideal case here would be to assemble a large dataset of public domain image files of scanned scores, each paired with human transcriptions, and to run all of the image files through multiple OMR applications. Image augmentation techniques could be used to add blur, noise, rotation, and skew to each of the images to generate multiple possible versions of the natural OMR errors added to each one. Even given a program to automate mouse clicks to operate a commercial OMR application, this process would not be foolproof. PhotoScore, for instance, occasionally makes errors in staff identification, which can lead to malformed MusicXML files or scores with the wrong number of instruments; the

program's workflow intends for the user to manually verify each page to ensure that staves are found in the correct positions and with the correct number of staff lines. In addition, some scans of musical scores include title pages with incipits (short excerpts of the main theme of the work), ornamentation that can be confused for musical notation, program notes, or re-engravings of each individual part of the score. Either humans must be present to be sure that only the correct pages of every scan are processed by the OMR application, or document analysis algorithms must be employed that can perform the same task with sufficient accuracy.

Finally, in the process of gathering more training data for this task, future researchers must contend with the challenge of standardizing "human-corrected transcriptions." As highlighted in Section 6.1.3 with Teresa Carreño's String Quartet and Section 6.2.2 with Fanny Hensel's String Quartet, human transcribers may intentionally diverge from the original engraving for editorial or stylistic reasons. When preparing guidelines for transcribers in the creation of the OpenScore String Quartets (OSSQ) corpus, Gotham et al. (2023) opted to fix any material they deemed "inconsistent," such as apparent editorial errors in printed editions of string quartets, with the intent that the finished symbolic transcriptions "make the best possible first impression on those who might bring them to the concert platform." This policy introduced slight but intentional discrepancies between the original score and the transcriptions that I here considered as ground truth.

For the purposes of using this data for OMR research, these deliberate deviations presented a significant challenge. During training, these discrepancies can inadvertently lead the model to focus on aspects of the score that are not genuine OMR errors, impairing its overall effectiveness when later used for inference. Assessing the impact of this issue on my method would necessitate extensive manual review and analysis of the training data. During the design and experimental phases of my research, this issue was not initially considered, and its significance only became apparent during the compilation of results. Future research in this area should explicitly account for these potential deviations and attempt to identify them in the training data ahead of time. This involves developing criteria or guidelines for what constitutes an acceptable human-corrected transcription and distinguishing between true errors and intentional editorial changes. Addressing this challenge is crucial for ensuring the reliability and accuracy of the training data, and by extension, the effectiveness of the machine-learning model in identifying genuine errors in musical transcriptions.

7.3.5 Application of the Method to Automatic Music Transcription

AMT refers to the process of computationally transcribing musical audio into symbolic scores. AMT is a difficult problem, and it has several similarities with the field of OMR. They both output symbolic music, their state-of-the-arts both use deep learning techniques, they both struggle with similar problems of training data availability, and they both often require a step of human review at the end of the process. In addition, they are both fields with significant applications in cataloging music and enabling musicological research.

Given a dataset of scores generated through AMT with accompanying corrected versions, it would be feasible to replace the OMR training data with AMT data and replicate the experiments in this dissertation using almost exactly the same methodology. However, the effectiveness of the OMR error detection workflow might not directly translate to AMT. Errors in AMT tend to be less overtly "un-musical" compared to those in OMR. For instance, a common AMT error involves misidentifying a note as its octave equivalent, particularly when the instrument's sound has a weak fundamental tone and a high proportion of its energy is in upper harmonics. This type of error, while significant, does not disrupt the musicality of a score as drastically as, say, an OMR error that replaces an accidental with a note head, resulting in a cluster chord. Therefore, while the methodologies developed for OMR error detection provide a valuable foundation, adapting them to the specific nuances and error profiles of AMT would require careful consideration and potentially significant modifications.

7.3.6 Human Evaluation of Time Saved by the Error Detector

The method for estimating the potential time saved by the error detector, developed in Section 3.4, is based on a highly simplified version of the OMR correction process. A better way to evaluate the usefulness of this error detection method would be to ask human transcribers to digitize some scores both with and without the error detector, and compare the time it takes in both cases. Though expensive and time-consuming, this kind of study would be necessary to definitively validate the method's usefulness.

A study by Thomae et al. (2022), that investigated the use of an algorithm to detect counterpoint errors to aid human transcribers in encoding of mensural scores, would be ideal to use as a blueprint for such a trial run. In this hypothetical trial, one or more expert musicians with experience transcribing music in a notation editor of their choice would be assigned to correct several excerpts of OMR output given the original image of each score. Half of the excerpts

would be raw OMR output, while the other half would be annotated with the results of the error detector. The elapsed time to complete each excerpt could be collected; in addition, software tools could be used to track their clicks and keystrokes, to see how the presence of the error annotations affects their workflow.

From these statistics and timings, researchers could create an evidence-based KLM model of the OMR correction process, collecting information on exactly how long different types of errors take to correct (see Section 3.2.2). This time-to-correct information would give a more interpretable measure of the method's performance than raw recall or R_{norm} scores. Integrated into the equations I derived in Section 6.1.3, this information would significantly improve the accuracy of the estimates of time saved. In addition, qualitative descriptions of what it is like to work with the error detector would be useful to construct better methods of visualization and new correction interfaces. Models could be re-trained after incorporating these new priorities into the loss function to better tailor the detector to human needs.

Bibliography

Abouzid, Houda, Otman Chakkor, Oscar Gabriel Reyes, and Sebastian Ventura. 2019. "Signal Speech Reconstruction and Noise Removal Using Convolutional Denoising Audioencoders with Neural Deep Learning." *Analog Integrated Circuits and Signal Processing* 100 (3): 501–512.

Aggarwal, Charu C. 2017. *Outlier Analysis*. Cham: Springer International Publishing.

Aguayo, Leonardo, and Guilherme A. Barreto. 2017. "Novelty Detection in Time Series Using Self-Organizing Neural Networks: A Comprehensive Evaluation." *Neural Processing Letters* 47:717–744.

Alfaro-Contreras, María, Jorge Calvo-Zaragoza, and José M. Iñesta. 2019. "Approaching End-to-End Optical Music Recognition for Homophonic Scores." In *Pattern Recognition and Image Analysis,* edited by Aythami Morales, Julian Fierrez, José Salvador Sánchez, and Bernardete Ribeiro, 11868:147–158. Cham: Springer International Publishing.

Alfaro-Contreras, María, Antonio Ríos-Vila, Jose J. Valero-Mas, José M. Iñesta, and Jorge Calvo-Zaragoza. 2022. "Decoupling Music Notation to Improve End-to-End Optical Music Recognition." *Pattern Recognition Letters* 158:157–163.

Alfaro-Contreras, María, David Rizo, Jose M Iñesta, and Jorge Calvo-Zaragoza. 2021. "OMR-Assisted Transcription: A Case Study with Early Prints." In *Proceedings of the International Society for Music Information Retrieval Conference.* Online.

Altschul, Stephen F. 1986. "Optimal Sequence Alignment Using Affine Gap Costs." *Bulletin of Mathematical Biology* 48 (5/6): 603–616.

———. 1991. "Amino Acid Substitution Matrices from an Information Theoretic Perspective." *Journal of Molecular Biology* 219 (3): 555–565.

Ambati, Bharat Ram, Mridul Gupta, Samar Husain, and Dipti Misra Sharma. 2010. "A High Recall Error Identification Tool for Hindi Treebank Validation." In *Proceedings of the International Conference on Language Resources and Evaluation,* 683–687. Valletta, Malta.

Angiulli, Fabrizio, and Fabio Fassetti. 2007. "Detecting Distance-Based Outliers in Streams of Data." In *Proceedings of the ACM conference on information and knowledge management.* Lisbon, Portugal: ACM Press.

Angiulli, Fabrizio, and Clara Pizzuti. 2005. "Outlier Mining in Large High-Dimensional Data Sets." *IEEE Transactions on Knowledge and Data Engineering* 17 (2): 203–215.

Baan, Joris, Maartje ter Hoeve, Marlies van der Wees, Anne Schuth, and Maarten de Rijke. 2019. "Do Transformer Attention Heads Provide Transparency in Abstractive Summarization?" In *Proceedings of the Workshop on Fairness, Accountability, Confidentiality, Transparency, and Safety in Information Retrieval.* Paris, France: ACM Press.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. "Neural Machine Translation by Jointly Learning to Align and Translate." *arXiv:1409.0473 [cs, stat].*

Bainbridge, David, and Tim Bell. 2001. "The Challenge of Optical Music Recognition." *Computers and the Humanities* 35 (2): 95–121.

Balke, Stefan, Sanu Pulimootil Achankunju, and Meinard Müller. 2015. "Matching Musical Themes Based on Noisy OCR and OMR Input." In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing,* 703–707.

Baró, Arnau, Pau Riba, and Alicia Fornés. 2022. "Musigraph: Optical Music Recognition Through Object Detection and Graph Neural Network." In *Frontiers in Handwriting Recognition,* edited by Utkarsh Porwal, Alicia Fornés, and Faisal Shafait, 171–184. Lecture Notes in Computer Science. Cham: Springer International Publishing.

Baro-Mas, Arnau. 2017. "Optical Music Recognition by Long Short-Term Memory Recurrent Neural Networks." Master's thesis, University of Alicante.

Basu, Sabyasachi, and Martin Meckesheimer. 2007. "Automatic Outlier Detection for Time Series: An Application to Sensor Data." *Knowledge and Information Systems* 11 (2): 137–154.

Bell, Samuel, Helen Yannakoudakis, and Marek Rei. 2019. "Context is Key: Grammatical Error Detection with Contextual Word Representations." In *Proceedings of the Workshop on Innovative Uses of NLP for Building Educational Applications,* 103–115. Florence, Italy: Association for Computational Linguistics.

Bellini, Pierfrancesco, Ivan Bruno, and Paolo Nesi. 2001. "Optical Music Sheet Segmentation." In *Proceedings of the International Conference on WEB Delivering of Music,* 183–190. Florence, Italy: IEEE Computer Society.

———. 2007. "Assessing Optical Music Recognition Tools." *Computer Music Journal* 31 (1): 68–93.

Bellman, R., and R. Kalaba. 1959. "On Adaptive Control Processes." *IRE Transactions on Automatic Control* 4 (2): 1–9.

Ben-Gal, Irad. 2005. "Outlier Detection." In *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers,* edited by Oded Maimon and Lior Rokach, 1–17. New York: Kluwer Academic Publishers.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. "Learning Long-Term Dependencies with Gradient Flow Is Difficult." *IEEE Transactions on Neural Networks* 5 (2): 157–167.

Berardinis, Jacopo de, Samuel Barrett, Angelo Cangelosi, and Eduardo Coutinho. 2020. "Modelling Long- and Short-Term Structure in Symbolic Music with Attention and Recurrence." In *Proceedings of the Joint Conference on AI Music Creativity.* Stockholm, Sweden.

Bhargave, Ojas. 2019. "Intelligent Error Detection and Advisory System for Practitioners of Music Using Audio Signal Processing and Computing." In *Proceedings of the International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering,* 257–260.

Bigdeli, Siavash Arjomand, and Matthias Zwicker. 2017. "Image Restoration using Autoencoding Priors." *arXiv:1703.09964 [cs.CV].*

Blázquez-García, Ane, Angel Conde, Usue Mori, and Jose A. Lozano. 2022. "A Review on Outlier/Anomaly Detection in Time Series Data." *ACM Computing Surveys* 54 (3): 1–33.

Blostein, Dorothea, and Henry S. Baird. 1992. "A Critical Survey of Music Image Analysis." In *Structured Document Image Analysis,* edited by Henry S. Baird, Horst Bunke, and Kazuhiko Yamamoto, 405–434. Berlin, Heidelberg: Springer.

Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent. 2013. "High-Dimensional Sequence Transduction." In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing,* 3178–3182. Vancouver, BC.

Breuel, Thomas M. 2015. "Benchmarking of LSTM Networks." *arXiv:1508.02774 [cs].*

Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. 2019. "Architecture." In *Deep Learning Techniques for Music Generation, Computational Synthesis and Creative Systems,* 41–90. Springer.

Brockett, Chris, William B. Dolan, and Michael Gamon. 2006. "Correcting ESL Errors Using Phrasal SMT Techniques." In *Proceedings of the International Conference on Computational Linguistics,* 249–256. Sydney, Australia.

Bronstein, Michael M., Joan Bruna, Taco Cohen, and Petar Veličković. 2021. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges.* arXiv:2104.13478 [cs].

Buckland, Michael, and Fredric Gey. 1994. "The Relationship Between Recall and Precision." *Journal of the American Society for Information Science* 45 (1): 12–19.

Bugge, Esben Paul, Kim Lundsteen Juncher, Brian Søborg Mathiasen, and Jakob Grue Simonsen. 2011. "Using Sequence Alignment and Voting to Improve Optical Music Recognition from Multiple Recognizers." In *Proceedings of the International Society for Music Information Retrieval Conference,* 405–410. Miami, FL.

Büyük, Osman, and Levent M. Arslan. 2021. "Learning from Mistakes: Improving Spelling Correction Performance with Automatic Generation of Realistic Misspellings." *Expert Systems* 38 (5).

Byrd, Donald, and Megan Schindele. 2006. "Prospects for Improving OMR with Multiple Recognizers." In *Proceedings of the International Society for Music Information Retrieval Conference.* Victoria, Canada.

Byrd, Donald, and Jakob Simonsen. 2015. "Towards a Standard Testbed for Optical Music Recognition: Definitions, Metrics, and Page Images." *Journal of New Music Research* 44:169–195.

Calvo-Zaragoza, Jorge, and Antonio-Javier Gallego. 2019. "A Selectional Auto-Encoder Approach for Document Image Binarization." *Pattern Recognition* 86:37–47.

Calvo-Zaragoza, Jorge, Jan Hajič Jr., and Alexander Pacha. 2021. "Understanding Optical Music Recognition." *ACM Computing Surveys* 53 (4): 1–35.

Calvo-Zaragoza, Jorge, and Jose Oncina. 2014. "Recognition of Pen-Based Music Notation: The HOMUS Dataset." In *International Conference on Pattern Recognition,* 3038–3043.

Calvo-Zaragoza, Jorge, and David Rizo. 2018a. "Camera-Primus: Neural End-to-End Optical Music Recognition on Realistic Monophonic Scores." In *Proceedings of the International Society for Music Information Retrieval Conference,* 248–255. Paris, France.

———. 2018b. "End-to-End Neural Optical Music Recognition of Monophonic Scores." *Applied Sciences* 8 (4): 606–629.

Calvo-Zaragoza, Jorge, Alejandro H. Toselli, and Enrique Vidal. 2019. "Handwritten Music Recognition for Mensural Notation with Convolutional Recurrent Neural Networks." *Pattern Recognition Letters* 128:115–121.

Calvo-Zaragoza, Jorge, Gabriel Vigliensoni, and Ichiro Fujinaga. 2017. "Pixel-wise Binarization of Musical Documents with Convolutional Neural Networks." In *Proceedings of the International Conference on Machine Vision Applications,* 362–365.

Canbek, Gürol, Seref Sagiroglu, Tugba Taskaya Temizel, and Nazife Baykal. 2017. "Binary Classification Performance Measures/Metrics: A Comprehensive Visualized Roadmap to Gain New Insights." In *Proceedings of the International Conference on Computer Science and Engineering,* 821–826.

Card, Stuart K., Thomas P. Moran, and Allen Newell. 1980. "The Keystroke-Level Model for User Performance Time with Interactive Systems." *Communications of the ACM* 23 (7): 396–410.

Castellanos, Francisco J., Jorge Calvo-Zaragoza, and José M. Iñesta. 2020. "A Neural Approach for Full-Page Optical Music Recognition of Mensural Documents." In *Proceedings of the International Society for Music Information Retrieval Conference,* 558–565. Montreal, Canada.

Castellanos, Francisco J., Carlos Garrido-Munoz, Antonio Ríos-Vila, and Jorge Calvo-Zaragoza. 2022. "Region-Based Layout Analysis of Music Score Images." *Expert Systems with Applications* 209:118211.

Caussinus, H., and A. Ruiz. 1990. "Interesting Projections of Multidimensional Data by Means of Generalized Principal Component Analyses." In *Compstat,* edited by Konstantin Momirović and Vesna Mildner, 121–126. Heidelberg: Physica.

Chalapathy, Raghavendra, and Sanjay Chawla. 2019. "Deep Learning for Anomaly Detection: A Survey." *arXiv:1901.03407 [cs, stat].*

Chen, Liang, Erik Stolterman, and Christopher Raphael. 2016. "Human-Interactive Optical Music Recognition." In *Proceedings of the International Society for Music Information Retrieval Conference.* New York City, NY.

Cheng, Jianpeng, Li Dong, and Mirella Lapata. 2016. "Long Short-Term Memory-Networks for Machine Reading." *arXiv:1601.06733 [cs].*

Cherry, Lorinda L., and William Vesterman. 1981. *Writing tools: The STYLE and DICTION programs.* Vol. 91. Computer Science Technical Reports. Murray Hill, NJ: Bell Labs.

Chicco, Davide, and Giuseppe Jurman. 2020. "The Advantages of the Matthews Correlation Coefficient (MCC) Over F1 Score and Accuracy in Binary Classification Evaluation." *BMC Genomics* 21 (6).

Child, Rewon, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. "Generating Long Sequences with Sparse Transformers." *arXiv:1904.10509 [cs.LG].*

Chou, Yi-Hui, I.-Chun Chen, Chin-Jui Chang, Joann Ching, and Yi-Hsuan Yang. 2021. *MidiBERT-Piano: Large-scale Pre-training for Symbolic Music Understanding.* arXiv:2107.05223 [cs.SD].

Church, Maura, and Michael Scott Cuthbert. 2014. "Improving Rhythmic Transcriptions Via Probability Models Applied Post-OMR." In *Proceedings of the International Society for Music Information Retrieval Conference,* 643–647. Taipei, Taiwan.

Civit, Miguel, Javier Civit-Masot, Francisco Cuadrado, and Maria J. Escalona. 2022. "A Systematic Review of Artificial Intelligence-Based Music Generation: Scope, Applications, and Future Trends." *Expert Systems with Applications* 209 (118190).

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter. 2016. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." In *Proceedings of the International Conference on Learning Representations.* San Juan, Puerto Rico.

Cogliati, Andrea, and Z. Duan. 2017. "A Metric for Music Notation Transcription Accuracy." In *Proceedings of the International Society for Music Information Retrieval Conference,* vol. 407-413. Suzhou, China.

Coüasnon, Bertrand, and Bernard Retif. 1995. "Using a Grammar for a Reliable Full Score Recognition System." In *Proceedings of the International Conference on Mathematics and Computing,* 187–196. Portland, OR.

Cover, T., and P. Hart. 1967. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory* 13 (1): 21–27.

Cuthbert, Michael Scott, and Christopher Ariza. 2010. "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data." In *Proceedings of the 11th International Society for Music Information Retrieval Conference,* 637–642. Utrecht, Netherlands.

Dai, Shuqi, Huiran Yu, and Roger B. Dannenberg. 2022. "What Is Missing in Deep Music Generation? A Study of Repetition and Structure in Popular Music." In *Proceedings of the 22nd International Society for Music Information Retrieval Conference.* Bengaluru, India.

Dai, Zihang, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context." In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* Florence, Italy.

Daigle, Alexandre. 2020. "Evaluation of Optical Music Recognition Software." Master's thesis, McGill University.

Damerau, Fred J. 1964. "A Technique for Computer Detection and Correction of Spelling Errors." *Communications of the ACM* 7 (3): 171–176.

de Reuse, Timothy, and Ichiro Fujinaga. 2022. "A Transformer-Based "Spellchecker" for Detecting Errors in OMR Output." In *Proceedings of the International Society for Music Information Retrieval Conference,* 789–796. Bengaluru, India.

deGroot-Maggetti, Jacob, Timothy de Reuse, Laurent Feisthauer, Samuel Howes, Yaolong Ju, Suzuka Kokobu, Sylvain Margot, Néstor Nápoles López, and Finn Upham. 2020. "Data Quality Matters: Iterative Corrections on a Corpus of Mendelssohn String Quartets and Implications for MIR Analysis." In *Proceedings of the International Society for Music Information Retrieval Conference,* 432–438. Montreal, Canada.

Dehghani, Mostafa, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019. "Universal Transformers." In *Proceedings of the International Conference on Learning Representations.* New Orleans, LA.

Delugach, Harry S., and Aldo de Moor. 2005. "Difference Graphs." In *Common Semantics for Sharing Knowledge: Proceedings of the International Conference on Conceptual Structures,* 41–53. Kassel, Germany: Kassel University Press.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." *arXiv:1810.04805 [cs].*

Droettboom, Michael, and Ichiro Fujinaga. 2004. "Symbol-Level Groundtruthing Environment for OMR." In *Proceedings of the International Symposium on Music Information Retrieval,* 497–500. Barcelona, Spain.

Droettboom, Michael, Ichiro Fujinaga, and Karl MacMillan. 2002. "Optical Music Interpretation," edited by G. Goos, J. Hartmanis, J. Van Leeuwen, Terry Caelli, Adnan Amin, Robert P. W. Duin, Dick De Ridder, and Mohamed Kamel, 2396:378–387. Berlin, Heidelberg: Springer Berlin Heidelberg.

Eck, Douglas, and Jüergen Schmidhuber. 2002. "Learning the Long-Term Structure of the Blues." In *Proceedings of the International Conference on Artificial Neural Networks,* 284–289. Springer Berlin Heidelberg.

Elkan, Charles, and Keith Noto. 2008. "Learning Classifiers from Only Positive and Unlabeled Data." In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 213–220. Las Vegas, NV: ACM Press.

Erkuş, Ekin Can, and Vilda Purutçuoğlu. 2021. "Outlier Detection and Quasi-Periodicity Optimization Algorithm: Frequency Domain Based Outlier Detection (FOD)." *European Journal of Operational Research* 291 (2): 560–574.

Ferguson, Thomas S. 1961. "On the Rejection of Outliers." In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability,* 1:235–287. Berkeley, CA.

Foscarin, Francesco, Florent Jacquemard, and Raphaël Fournier-S'niehotta. 2019. "A Diff Procedure for Music Score Files." In *Proceedings of the International Conference on Digital Libraries for Musicology,* 58–64. The Hague, Netherlands: ACM.

Foster, Jennifer, and Øistein E. Andersen. 2009. "GenERRate: Generating Errors for Use in Grammatical Error Detection." In *Proceedings of the Workshop on Innovative Use of NLP for Building Educational Applications,* 82–90. Boulder, CO: Association for Computational Linguistics.

Freeman, Elizabeth A., and Gretchen G. Moisen. 2008. "A Comparison of the Performance of Threshold Criteria for Binary Classification in Terms of Predicted Prevalence and Kappa." *Ecological Modelling* 217 (1): 48–58.

Frisch, Walter. 1982. "Brahms, Developing Variation, and the Schoenberg Critical Tradition." *19th-Century Music* 5 (3): 215–232.

Fujinaga, Ichiro. 1988. "Optical Music Recognition Using Projections." Master's thesis, McGill University.

———. 1999. "Exemplar-Based Learning in Adaptive Optical Music Recognition System." In *Proceedings of the International Conference on Mathematics and Computing.*

Gao, Jing, Haibin Cheng, and Pang-Ning Tan. 2006. "Semi-Supervised Outlier Detection." In *Proceedings of the ACM Symposium on Applied Computing.* Dijon, France: ACM Press.

Ge, Rong, Sham M. Kakade, Rahul Kidambi, and Praneeth Netrapalli. 2019. "Rethinking Learning Rate Schedules for Stochastic Optimization." In *Proceedings of the International Conference on Learning Representations.* New Orleans, LA.

Ge, Tao, Furu Wei, and Ming Zhou. 2018. "Reaching Human-level Performance in Automatic Grammatical Error Correction: An Empirical Study." *arXiv:1807.01270 [cs].*

Gehring, Jonas, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. "Convolutional Sequence to Sequence Learning." In *Proceedings of the International Conference on Machine Learning,* 2029–2043. Sydney, Australia.

Godsill, Simon J., Patrick J. Wolfe, and William N.W. Fong. 2001. "Statistical Model-Based Approaches to Audio Restoration and Analysis." *Journal of New Music Research* 30 (4): 323–338.

Goecke, Roland. 2006. "Building a System for Writer Identification on Handwritten Music Scores." In *Proceedings of the International Conference on Pattern Recognition,* 803–806. Hong Kong, China.

Gotham, Mark, Maureen Redbond, Bruno Bower, and Peter Jonas. 2023. "The "OpenScore String Quartet" Corpus." In *Proceedings of the 10th International Conference on Digital Libraries for Musicology,* 49–57. DLfM '23. New York, NY, USA: Association for Computing Machinery.

Greaves-Tunnell, Alexander, and Zaid Harchaoui. 2019. "A Statistical Investigation of Long Memory in Language and Music." *arXiv:1904.03834 [cs, eess, stat].*

Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2017. "LSTM: A Search Space Odyssey." *IEEE Transactions on Neural Networks and Learning Systems* 28 (10): 2222–2232.

Habrard, Amaury, José Manuel Iñesta, David Rizo, and Marc Sebban. 2008. "Melody Recognition with Learned Edit Distances." In *Structural, Syntactic, and Statistical Pattern Recognition,* edited by Niels Da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog, 5342:86–96. Springer Berlin Heidelberg.

Hajic, Jan, Jiri Novotný, Pavel Pecina, and Jaroslav Pokorný. 2016. "Further Steps Towards a Standard Testbed for Optical Music Recognition." In *Proceedings of the International Society for Music Information Retrieval Conference,* 157–162. New York, NY.

Hajič, Jan. 2018. "A Case for Intrinsic Evaluation of Optical Music Recognition." In *Proceedings of the Workshop on Reading Music Systems,* 14–15. Paris, France.

Hajič, Jan, and Pavel Pecina. 2017. "The MUSCIMA++ Dataset for Handwritten Optical Music Recognition." In *International Conference on Document Analysis and Recognition,* 39–46.

Hamming, R. W. 1950. "Error Detecting and Error Correcting Codes." *The Bell System Technical Journal* 29 (2): 147–160.

Hankinson, Andrew Noah. 2015. "Optical Music Recognition Infrastructure for Large-Scale Music Document Analysis." Doctoral Thesis, McGill University.

Hanley, J. A., and B. J. McNeil. 1982. "The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve." *Radiology* 143 (1): 29–36.

Hawkins, D. M. 1980. *Identification of Outliers.* Dordrecht: Springer Netherlands.

Hawthorne, Curtis, Anna Huang, Daphne Ippolito, and Douglas Eck. 2018. "Transformer-NADE for Piano Performances." In *Proceedings of the Conference on Neural Information Processing Systems.* Montreal, Canada.

Hawthorne, Curtis, Ian Simon, Rigel Swavely, Ethan Manilow, and Jesse Engel. 2021. "Sequence-to-Sequence Piano Transcription with Transformers." In *Proceedings of the International Society for Music Information Retrieval Conference,* 246–253. Online.

Helsen, Kate, Jennifer Bain, Ichiro Fujinaga, Andrew Hankinson, and Debra Lacoste. 2014. "Optical Music Recognition and Manuscript Chant Sources." *Early Music* 42 (4): 555–558.

Henikoff, Steven, and Jorja G. Henikoff. 1992. "Amino Acid Substitution Matrices from Protein Blocks." *Proceedings of the National Academy of Sciences of the United States of America* 89 (22): 10915–10919.

Hinton, G. E., and R. R. Salakhutdinov. 2006. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313 (5786): 504–507.

Hochreiter, Sepp. 1998. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06 (02): 107–116.

Hochreiter, Sepp, Yoshua Bengio, Paolo Frasconi, and Jurgen Schmidhuber. 2001. "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies." In *A Field Guide to Dynamical Recurrent Networks,* edited by S. C. Kremer and J. F. Kolen, 237–243. IEEE.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997a. "Long Short-Term Memory." *Neural Computation* 9:1735–1780.

———. 1997b. "LSTM Can Solve Hard Long Time Lag Problems." *Advances in Neural Information Processing Systems,* 473–479.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5): 359–366.

Htut, Phu Mon, and Joel Tetreault. 2019. "The Unbearable Weight of Generating Artificial Errors for Grammatical Error Correction." *arXiv:1907.08889 [cs].*

Huang, Anna, Ashish Vaswani Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew Dai, Matthew Hoffman, and Douglas Eck. 2019. "Music Transformer: Generating Music with Long-Term Structure." In *Proceedings of the International Conference on Learning Representations.* New Orleans, LA.

Jagadish, H., V. Nick Coudas, and S Muthukrishnan. 1999. "Mining Deviants in a Time Series Database." In *Proceedings of the International Conference on Very Large Databases,* 102–113.

Jhamtani, Harsh, and Taylor Berg-Kirkpatrick. 2019. "Modeling Self-Repetition in Music Generation Using Generative Adversarial Networks." In *Proceedings of the 36th International Conference on Machine Learning.* Long Beach, CA.

Ji, Shulei, Xinyu Yang, and Jing Luo. 2023. "A Survey on Deep Learning for Symbolic Music Generation: Representations, Algorithms, Evaluations, and Challenges." *ACM Computing Surveys* 56 (1): 1–39.

Jin, Rong, and Christopher Raphael. 2012. "Interpreting Rhythm in Optical Music Recognition." In *Proceedings of the International Society for Music Information Retrieval Conference,* 151–156. Porto, Portugal.

Jones, Graham, Bee Ong, Ivan Bruno, and Kia Ng. 2008. "Optical Music Imaging: Music Document Digitisation, Recognition, Evaluation, and Restoration." In *Interactive Multimedia Music Technologies,* edited by Kia Ng and Paolo Nesi, 50–79. IGI Global.

Karystinaios, Emmanouil, and Gerhard Widmer. 2022. "Cadence Detection in Symbolic Classical Music using Graph Neural Networks." In *Proceedings of the International Society for Music Information Retrieval Conference,* 917–926. Bengaluru, India: arXiv.

Kassler, Michael. 1972. "Optical Character-Recognition of Printed Music: A Review of Two Dissertations." Edited by Dennis Howard Pruslin and David Stewart Prerau. *Perspectives of New Music* 11 (1): 250–254.

Katharopoulos, Angelos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention." In *Proceedings of the International Conference on Machine Learning.* Online.

Kato, Hirokazu, and Seiji Inokuchi. 1992. "A Recognition System for Printed Piano Music Using Musical Knowledge and Constraints." In *Structured Document Image Analysis,* edited by Henry S. Baird, Horst Bunke, and Kazuhiko Yamamoto, 435–455. Berlin, Heidelberg: Springer.

Kieu, Tung, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2019. "Outlier Detection for Time Series with Recurrent Autoencoder Ensembles." In *Proceedings of the International Joint Conference on Artificial Intelligence,* 2725–2732. Macao, China.

Kieu, Tung, Bin Yang, and Christian S. Jensen. 2018. "Outlier Detection for Multidimensional Time Series Using Deep Neural Networks." In *Proceedings of the IEEE International Conference on Mobile Data Management,* 125–134.

Kingma, Diederik P., and Jimmy Ba. 2017. "Adam: A Method for Stochastic Optimization." In *Proceedings of the International Conference on Learning Representations.* Toulon, France.

Kitaev, Nikita, Łukasz Kaiser, and Anselm Levskaya. 2020. "Reformer: The Efficient Transformer." In *Proceedings of the International Conference on Learning Representations.* Online.

Kotsiantis, S.B. 2007. "Supervised Machine Learning: A Review of Classification Techniques." *Informatica* 31:249–268.

Lam, Siu Kwan, Antoine Pitrou, and Stanley Seibert. 2015. "Numba: A LLVM-Based Python JIT Compiler." In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in High-Performance Computing.* Austin, TX: ACM.

Lamont, Alexandra, and Nicola Dibben. 2001. "Motivic structure and the perception of similarity." *Music Perception* 18 (3): 245–274.

Lee, Juho, Yoonho Lee, Jungtaek Kim, Adam R Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks." In *Proceedings of the International Conference on Machine Learning.* Long Beach, CA.

Lee-Thorp, James, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. 2022. "FNet: Mixing Tokens with Fourier Transforms." In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics.* Seattle, WA.

Lemström, Kjell, and Anna Pienimäki. 2007. "On Comparing Edit Distance and Geometric Frameworks in Content-Based Retrieval of Symbolically Encoded Polyphonic Music." *Musicae Scientiae* 11 (1_suppl): 135–152.

Leshno, Moshe, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function." *Neural Networks* 6 (6): 861–867.

Levenshtein, Vladimir I. 1966. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Soviet Physics Doklady* 10 (8).

Li, Mu, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. 2014. "Efficient Mini-Batch Training for Stochastic Optimization." In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 661–670. New York, NY: ACM Press.

Li, Ying, Binbin Fan, Weiping Zhang, and Zhiqiang Jiang. 2021. "TireNet: A High Recall Rate Method for Practical Application of Tire Defect Type Classification." *Future Generation Computer Systems* 125:1–9.

Likforman-Sulem, Laurence, Abderrazak Zahour, and Bruno Taconet. 2007. "Text Line Segmentation of Historical Documents: A Survey." *International Journal of Document Analysis and Recognition* 9 (2-4): 123–138.

Lipton, Zachary C., Charles Elkan, and Balakrishnan Naryanaswamy. 2014. "Optimal Thresholding of Classifiers to Maximize F1 Measure." In *Proceedings of Machine Learning and Knowledge Discovery in Databases,* 8725:225–239. Dublin, Ireland.

Liu, Wenyin, Liang Zhang, Long Tang, and Dov Dori. 1999. "Cost Evaluation of Interactively Correcting Recognized Engineering Drawings." In *Proceedings of the IAPR Workshop on Graphics Recognition,* 329–334. Berlin, Heidelberg.

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. 2015. "Effective Approaches to Attention-Based Neural Machine Translation." *arXiv:1508.04025 [cs].*

Macdonald, N., L. Frase, P. Gingrich, and S. Keenan. 1982. "The Writer's Workbench: Computer Aids for Text Analysis." *IEEE Transactions on Communications* 30 (1): 105–110.

Madell, Jaime, and Sylvie Hébert. 2008. "Eye Movements and Music Reading: Where Do We Look Next?" *Music Perception* 26 (2): 157–170.

Madi, Nora, and Hend S. Al-Khalifa. 2018. "Grammatical Error Checking Systems: A Review of Approaches and Emerging Directions." In *Proceedings of the International Conference on Digital Information Management,* 142–147. Berlin, Germany: IEEE.

Magdy, Walid, and Gareth J.F. Jones. 2010. "PRES: A Score Metric for Evaluating Recall-Oriented Information Retrieval Applications." In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval,* 611–618. Geneva, Switzerland: ACM Press.

Mahalanobis, Prasanta Chandra. 1936. "On the Generalized Distance in Statistics." *Proceedings of the National Institute of Sciences of India* 2 (1): 49–55.

Malhotra, Pankaj, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. "Long Short-Term Memory Networks for Anomaly Detection in Time Series." In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning,* 89–95. Bruges, Belgium.

Margulis, Elizabeth Hellmuth. 2014. *On Repeat: How Music Plays the Mind.* Oxford University Press.

McLeod, Andrew, James Owers, and Kazuyoshi Yoshii. 2020. "The MIDI Degradation Toolkit: Symbolic Music Augmentation and Correction." *arXiv:2010.00059 [cs, eess].*

Medeot, Gabriele, Srikanth Cherla, Katerina Kosta, Matt McVicar, Samer Abdallah, and Marco Selvi. 2018. "StructureNet: Inducing Structure in Generated Melodies." In *Proceedings of the International Society for Music Information Retrieval Conference,* 725–731. Paris, France.

Al-Megren, Shiroq, Joharah Khabti, and Hend S. Al-Khalifa. 2018. "A Systematic Review of Modifications and Validation Methods for the Extension of the Keystroke-Level Model." *Advances in Human-Computer Interaction* 2018:1–26.

Mengarelli, Luciano, Bruno Kostiuk, João G. Vitório, Maicon A. Tibola, William Wolff, and Carlos N. Silla. 2020. "OMR Metrics and Evaluation: A Systematic Review." *Multimedia Tools and Applications* 79 (9): 6383–6408.

Middleton, Richard. 1983. "'Play it Again Sam': Some Notes on the Productivity of Repetition in Popular Music." *Popular Music* 3:235–270.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. *Efficient Estimation of Word Representations in Vector Space.* arXiv:1301.3781.

Miyao, Hidetoshi, and Masayuki Okamoto. 2004. "Stave Extraction for Printed Music Scores Using DP Matching." *Journal of Advanced Computational Intelligence and Intelligent Informatics* 8 (2): 208–215.

Moffat, Alistair, and Justin Zobel. 2008. "Rank-Biased Precision for Measurement of Retrieval Effectiveness." *ACM Transactions on Information Systems* 27 (1): 1–27.

Moon, Sangwhan, and Naoaki Okazaki. 2021. "Effects and Mitigation of Out-of-vocabulary in Universal Language Models." *Journal of Information Processing* 29:490–503.

Mozer, Michael C. 1991. "Induction of Multiscale Temporal Structure." *Advances in Neural Information Processing Systems* 4:276–281.

———. 1994. "Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multiscale Processing." *Cognitive Science* 6:247–280.

Munir, Mohsin, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. 2019. "DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series." *IEEE Access* 7:1991–2005.

Mustafa, Wan Azani, and Mohamed Mydin M. Abdul Kader. 2018. "Binarization of Document Images: A Comprehensive Review." *Journal of Physics: Conference Series* 1019 (1): 012023.

Naber, Daniel. 2003. "A Rule-Based Style and Grammar Checker." Doctoral Thesis, Universität Bielefeld.

Nagy, G. 1995. "Document Image Analysis: Automated Performance Evaluation." In *Document Image Analysis Systems,* 137–156. Singapore: World Scientific Publishing CO.

Nakamura, Eita, Kazuyoshi Yoshii, and Haruhiro Katayose. 2017. "Performance Error Detection and Post-Processing for Fast and Accurate Symbolic Music Alignment." In *Proceedings of the International Society for Music Information Retrieval Conference,* 347–353. Suzhou, China.

Navarro, Gonzalo. 2001. "A Guided Tour to Approximate String Matching." *ACM Computing Surveys* 33 (1): 31–88.

Needleman, Saul B., and Christian D. Wunsch. 1970. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins." *Journal of Molecular Biology* 48 (3): 443–453.

Neuwirth, Markus, Daniel Harasim, Fabian C. Moss, and Martin Rohrmeier. 2018. "The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets." *Frontiers in Digital Humanities* 5 (16).

Ng, Kia, and Andrew Jones. 2003. "A Quick-Test for Optical Music Recognition Systems." In *Proceedings of the MUSICNETWORK Open Workshop on Optical Music Recognition System.* Leeds, United Kingdom.

Novotný, Jirı, and Jaroslav Pokorný. 2015. "Introduction to Optical Music Recognition: Overview and Practical Challenges." In *Proceedings of the Annual International Workshop on Databases, Texts, Specifications, and Objects,* 65–76. Jičín, Czech Republic.

Nowakowski, Matthias, and Aristotelis Hadjakos. 2023. "Estimating Interaction Time in Music Notation Editors." In *Proceedings of the International Symposium on Computer Music Multidisciplinary Research,* 335–346. Tokyo, Japan.

Obafemi-Ajayi, Tayo, and Gady Agam. 2013. "Goal-Oriented Evaluation of Binarization Algorithms for Historical Document Images." In *Proceedings in Document Recognition and Retrieval,* 8658:265–274. Burlingame, CA.

Ozkan, Huseyin, Fatih Ozkan, and Suleyman S. Kozat. 2016. "Online Anomaly Detection Under Markov Statistics With Controllable Type-I Error." *IEEE Transactions on Signal Processing* 64 (6): 1435–1445.

Pacha, Alexander, and Jorge Calvo-Zaragoza. 2018. "Optical Music Recognition in Mensural Notation with Region-Based Convolutional Neural Networks." In *Proceedings of the International Society for Music Information Retrieval Conference,* 240–248. Paris, France.

———. 2019. "Learning Notation Graph Construction for Full-Pipeline Optical Music Recognition." In *Proceedings of the International Society for Music Information Retrieval Conference,* 75–83. Delft, Netherlands.

Pacha, Alexander, and Horst Eidenberger. 2017. "Towards a Universal Music Symbol Classifier." In *Proceedings of the IAPR International Conference on Document Analysis and Recognition,* 2:35–36.

Padilla, Victor, Alan Marsden, Alex McLean, and Kia Ng. 2014. "Improving OMR for Digital Music Libraries with Multiple Recognisers and Multiple Sources." In *Proceedings of the International Society for Music Information Retrieval Conference,* 517–523. New York, NY.

Papadimitriou, Spiros, Jimeng Sun, and Christos Faloutsos. 2005. "Streaming Pattern Discovery in Multiple Time-Series." In *Proceedings of the Very Large Databases Conference.* Trondheim, Norway.

Parmar, Niki, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. "Image Transformer." *arXiv:1802.05751 [cs].*

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. "Automatic Differentiation in Pytorch." In *Proceedings of the Conference on Neural Information Processing Systems.* Long Beach, CA.

Penny, Kay I. 1996. "Appropriate Critical Values When Testing for a Single Multivariate Outlier by Using the Mahalanobis Distance." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 45 (1): 73–81.

Pham, Ngoc-Quan, Thanh-Le Ha, Tuan-Nam Nguyen, Thai-Son Nguyen, Elizabeth Salesky, Sebastian Stueker, Jan Niehues, and Alexander Waibel. 2020. "Relative Positional Encoding for Speech Recognition and Direct Translation." In *Proceedings of Interspeech.* Online.

Powers, David M. W. 2020. "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation." *arXiv:2010.16061.*

Prerau, David S. 1975. "DO-RE-MI: A Program That Recognizes Music Notation." *Computers and the Humanities* 9 (1): 25–29.

Prost, Jean-Philippe. 2009. "Grammar Error Detection with Best Approximated Parse." In *Proceedings of the International Conference on Parsing Technologies,* 172–179. Paris, France: Association for Computational Linguistics.

Pruslin, Dennis Howard. 1966. "Automated Recognition of Sheet Music." Doctoral Thesis, Massachusetts Institute of Technology.

Pugin, Laurent. 2006. "Optical Music Recognition of Early Typographic Prints using Hidden Markov Models." In *Proceedings of the International Society for Music Information Retrieval Conference.* Victoria, Canada.

———. 2015. "The Challenge of Data in Digital Musicology." *Frontiers in Digital Humanities* 2 (4): 1–3.

Pugin, Laurent, John Ashley Burgoyne, and Ichiro Fujinaga. 2007. "Reducing Costs for Digitising Early Music with Dynamic Adaptation." In *Research and Advanced Technology for Digital Libraries,* edited by László Kovács, Norbert Fuhr, and Carlo Meghini, 4675:471–474. Berlin: Springer Berlin Heidelberg.

Pugin, Laurent, Rodolfo Zitellini, and Perry Roland. 2014. "Verovio: A Library for Engraving Mei Music Notation into Svg." In *Proceedings of the International Society for Music Information Retrieval Conference,* 107–112. Taipei, Taiwan.

Ranjan, Nihar, Kaushal Mundada, Kunal Phaltane, and Saim Ahmad. 2016. "A Survey on Techniques in NLP." *International Journal of Computer Applications* 134 (8): 6–9.

Rastall, Richard. 1982. *The Notation of Western Music: An Introduction.* New York, NY: St. Martin's Press.

Rebelo, Ana, Ichiro Fujinaga, Filipe Paszkiewicz, Andre R. S. Marcal, Carlos Guedes, and Jaime S. Cardoso. 2012. "Optical Music Recognition: State-of-the-Art and Open Issues." *International Journal of Multimedia Information Retrieval* 1 (3): 173–190.

Reed, Todd K., and James R. Parker. 1996. "Automatic Computer Recognition of Printed Music." In *Proceedings of the International Conference on Pattern Recognition,* 3:803–807.

Rei, Marek, Mariano Felice, Zheng Yuan, and Ted Briscoe. 2017. "Artificial Error Generation with Machine Translation and Syntactic Patterns." *arXiv:1707.05236 [cs].*

Rice, Stephen V., Frank R. Jenkins, and Thomas A. Nartker. 1995. *The Fourth Annual Test of OCR Accuracy.* Technical Report 95-03. Las Vegas, NV: University of Nevada.

Ríos-Vila, Antonio, Jorge Calvo-Zaragoza, and David Rizo. 2020. "Evaluating Simultaneous Recognition and Encoding for Optical Music Recognition." In *Proceedings of the International Conference on Digital Libraries for Musicology,* 10–17. Montreal, Canada: ACM.

Ríos-Vila, Antonio, Miquel Esplà-Gomis, David Rizo, Pedro J. Ponce de León, and José M. Iñesta. 2021. "Applying Automatic Translation for Optical Music Recognition's Encoding Step." *Applied Sciences* 11 (9): 3890.

Ríos-Vila, Antonio, David Rizo, José M. Iñesta, and Jorge Calvo-Zaragoza. 2023. "End-to-End Optical Music Recognition for Pianoform Sheet Music." *International Journal on Document Analysis and Recognition* 26:347–362.

Rizo, David, Jorge Calvo-Zaragoza, and José M. Iñesta. 2018. "MuRET: A Music Recognition, Encoding, and Transcription Tool." In *Proceedings of the International Conference on Digital Libraries for Musicology,* 52–56. New York, NY: ACM.

Robbins, Herbert, and Sutton Monro. 1951. "A Stochastic Approximation Method." *The Annals of Mathematical Statistics* 22 (3): 400–407.

Roberts, Teresa L., and Thomas P. Moran. 1983. "The Evaluation of Text Editors: Methodology and Empirical Results." *Communications of the ACM* 26 (4): 265–283.

Robertson, Stephen. 2008. "A New Interpretation of Average Precision." In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* 689–690. New York, NY: ACM Press.

Rocchio, Joseph. 1964. "Performance Indices for Document Retrieval Systems." In *Information Storage and Retrieval.* Cambridge, MA: Computation Laboratory of Harvard University.

Rojas, Raúl. 1996. "The Backpropagation Algorithm." In *Neural Networks,* 149–182. Berlin, Heidelberg: Springer Berlin Heidelberg.

Rossant, Florence, and Isabelle Bloch. 2006. "Robust and Adaptive OMR System Including Fuzzy Modeling, Fusion of Musical Rules, and Possible Error Detection." *EURASIP Journal on Advances in Signal Processing* 2007 (1).

Rozovskaya, Alla, and Dan Roth. 2010. "Training Paradigms for Correcting Errors in Grammar and Usage." In *Human Language Technologies: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics.* Los Angeles, CA.

Ruder, Sebastian. 2017. "An Overview of Gradient Descent Optimization Algorithms." *arXiv:1609.04747 [cs.LG].*

Sakurada, Mayu, and Takehisa Yairi. 2014. "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction." In *Proceedings of the Workshop on Machine Learning for Sensory Data Analysis,* 4–11. Gold Coast, Australia: ACM Press.

Sands, Janelle C. 2020. "Efficient Optical Music Recognition Validation Using MIDI Sequence Data." Master's thesis, Massachusetts Institute of Technology.

Sapp, Craig. 2013. "OMR Comparison of SmartScore and SharpEye." Accessed August 12, 2023. ccrma.stanford.edu/~craig/mro-compare-beethoven/.

Sauro, Jeff. 2009. "Estimating Productivity: Composite Operators for Keystroke Level Modeling." In *Human-Computer Interaction: New Trends,* edited by Julie A. Jacko, 5610:352–361. Berlin, Heidelberg: Springer.

Scholz, Ricardo, Geber Ramalho, and Giordano Cabral. 2016. "Cross Task Study on MIREX Recent Results: An Index for Evolution Measurement and Some Stagnation Hypotheses." In *Proceedings of the International Society for Music Information Retrieval Conference,* 372–380. New York City, NY.

Schubert, Erich, Arthur Zimek, and Hans-Peter Kriegel. 2014. "Local Outlier Detection Reconsidered: A Generalized View on Locality with Applications to Spatial, Video, and Network Outlier Detection." *Data Mining and Knowledge Discovery* 28 (1): 190–237.

Shaalan, Khaled F. 2005. "Arabic GramCheck: A Grammar Checker for Arabic." *Software: Practice and Experience* 35 (7): 643–665.

Shalaby, Walid, and Wlodek Zadrozny. 2019. "Patent Retrieval: A Literature Review." *Knowledge and Information Systems* 61 (2): 631–660.

Shatri, Elona, and György Fazekas. 2020. "Optical Music Recognition: State of the Art and Major Challenges." *arXiv:2006.07885 [cs.CV].*

Shaw, Peter, Jakob Uszkoreit, and Ashish Vaswani. 2018. "Self-Attention with Relative Position Representations." *arXiv:1803.02155 [cs].*

Shen, Zhuoran, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. 2021. "Efficient Attention: Attention with Linear Complexities." In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision,* 3531–3539.

Sidorov, Kirill, Andrew Jones, and David Marshall. 2014. "Music Analysis as a Smallest Grammar Problem." In *Proceedings of the International Society for Music Information Retrieval Conference,* 301–306. Taipei, Taiwan.

Smith, Leslie N. 2017. "Cyclical Learning Rates for Training Neural Networks." In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.*

Smith, Leslie N., and Nicholay Topin. 2018. "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates." *arXiv:1708.07120 [cs.LG].*

Smith, T.F., and M.S. Waterman. 1981. "Identification of Common Molecular Subsequences." *Journal of Molecular Biology* 147 (1): 195–197.

Sofaer, Helen R., Jennifer A. Hoeting, and Catherine S. Jarnevich. 2019. "The Area Under the Precision-Recall Curve as a Performance Metric for Rare Binary Events." *Methods in Ecology and Evolution* 10 (4): 565–577.

Stamatopoulos, Nikolaos, Georgios Louloudis, and Basilis Gatos. 2015. "Goal-Oriented Performance Evaluation Methodology for Page Segmentation Techniques." In *International Conference on Document Analysis and Recognition,* 281–285.

Staudemeyer, Ralf C., and Eric R. Morris. 2019. "Understanding LSTM – A Tutorial into Long Short-Term Memory Recurrent Neural Networks." *arXiv:1909.09586.*

Stehouwer, Herman, and Menno van Zaanen. 2009. "Language Models for Contextual Error Detection and Correction." In *Proceedings of the EACL Workshop on Computational Linguistic Aspects of Grammatical Inference,* 41–48. Athens, Greece.

Stone, Kurt. 1980. *Music Notation in the Twentieth Century: A Practical Guidebook.* First Edition. New York, NY: W. W. Norton.

Strayer, Hope. 2013. "From Neumes to Notes: The Evolution of Music Notation." *Musical Offerings* 4 (1): 1–14.

Student. 1908. "The Probable Error of a Mean." *Biometrika* 6 (1): 1–25.

Sturm, Bob L., João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. 2016. "Music Transcription Modelling and Composition Using Deep Learning." In *Proceedings of the 2nd Conference on Computer Simulation of Musical Creativity.*

Subramaniam, S., T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. 2008. "Online Outlier Detection in Sensor Data Using Non-Parametric Models." In *Proceedings of the International Conference on Very Large Databases,* 187–198.

Sulaiman, Alaa, Khairuddin Omar, and Mohammad F. Nasrudin. 2019. "Degraded Historical Document Binarization: A Review on Issues, Challenges, Techniques, and Future Directions." *Journal of Imaging* 5 (4): 48.

Tay, Yi, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020. "Synthesizer: Rethinking Self-Attention in Transformer Models." In *Proceedings of the International Conference on Machine Learning.* Online.

Tay, Yi, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. "Long Range Arena: A Benchmark for Efficient Transformers." In *Proceedings of the International Conference on Learning Representations.* Online.

Tay, Yi, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2023. "Efficient Transformers: A Survey." *ACM Computing Surveys* 55 (6): 1–28.

Tetreault, Joel R., and Martin Chodorow. 2008. "The Ups and Downs of Preposition Error Detection in ESL Writing." In *Proceedings of the 22nd International Conference on Computational Linguistics,* 1:865–872. Manchester, UK: Association for Computational Linguistics.

Thomae, Martha E. 2023. "Guatemalan Cathedral Choirbooks: From Manuscript to Digital Images to Edited Symbolic Scores." Doctoral Thesis, McGill University.

Thomae, Martha E., Julie E. Cumming, and Ichiro Fujinaga. 2022. "Counterpoint Error-Detection Tools for Optical Music Recognition of Renaissance Polyphonic Music." In *Proceedings of the International Society for Music Information Retrieval Conference,* 501–508. Bengaluru, India.

Thomae, Martha E., David Rizo, Antonio Ríos-Vila, José M. Iñesta, and Jorge Calvo-Zaragoza. 2020. "Retrieving Music Semantics from Optical Music Recognition by Machine Translation." In *Proceedings of the Music Encoding Conference,* 19–25. Online.

Trier, O.D., and A.K. Jain. 1995. "Goal-Directed Evaluation of Binarization Methods." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17 (12): 1191–1201.

Tuggener, Lukas, Ismail Elezi, Jürgen Schmidhuber, Marcello Pelillo, and Thilo Stadelmann. 2018. "DeepScores: A Dataset for Segmentation, Detection and Classification of Tiny Objects." *arXiv:1804.00525 [cs.CV].*

Ukkonen, Esko. 1985. "Algorithms for Approximate String Matching." *Information and Control* 64 (1-3): 100–118.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." In *Proceedings of the Conference on Neural Information Processing Systems.* Long Beach, CA.

Werbos, Paul J. 1990. "Backpropagation Through Time: What It Does and How to Do It." *Proceedings of the IEEE* 78 (10): 1550–1560.

White, Max, and Alla Rozovskaya. 2020. "A Comparative Study of Synthetic Data Generation Methods for Grammatical Error Correction." In *Proceedings of the Workshop on Innovative Use of NLP for Building Educational Applications,* 198–208. Seattle, WA: Association for Computational Linguistics.

Xia, Yan, Xudong Cao, Fang Wen, Gang Hua, and Jian Sun. 2015. "Learning Discriminative Reconstructions for Unsupervised Outlier Removal." In *Proceedings of the 2015 IEEE International Conference on Computer Vision,* 1511–1519. Santiago, Chile: IEEE.

Xu, Kelvin, Jimmy Lei, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. 2015. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." In *Proceedings of the International Conference on Machine Learning.* Lille, France.

Ycart, Adrien, Daniel Stoller, and Emmanouil Benetos. 2019. "A Comparative Study of Neural Models for Polyphonic Music Sequence Transduction." In *Proceedings of the International Society for Music Information Retrieval Conference,* 470–477. Delft, Netherlands.

Yu, Yong, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures." *Neural Computation* 31 (7): 1235–1270.

Yu, Yufeng, Yuelong Zhu, Shijin Li, and Dingsheng Wan. 2014. "Time Series Outlier Detection Based on Sliding Window Prediction." *Mathematical Problems in Engineering* 2014:1–14.

Yuan, Zheng, and Ted Briscoe. 2016. "Grammatical Error Correction Using Neural Machine Translation." In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* 380–386. San Diego, CA: Association for Computational Linguistics.

Zhang, Emily H. 2017. "An Efficient Score Alignment Algorithm and Its Applications." Master's thesis, Massachusetts Institute of Technology.

Zhang, Haotian, Mustafa Abualsaud, Nimesh Ghelani, Mark D. Smucker, Gordon V. Cormack, and Maura R. Grossman. 2018. "Effective User Interaction for High-Recall Retrieval: Less is More." In *Proceedings of the ACM International Conference on Information and Knowledge Management,* 187–196. Torino, Italy.

Zhao, Jingyu, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. 2020. "Do RNN and LSTM have Long Memory?" In *Proceedings of the International Conference on Machine Learning.* Online.