GitHub Events Through the Lens of Knowledge Graphs

Kian Ahrabian

School of Computer Science
McGill University
Montreal, Quebec, Canada

December 2020

A thesis submitted to McGill University in partial
fulfillment of the requirements of the degree of
Master of Science

Abstract

In the past few years, GitHub has become the leading platform for open-source software development, offering many collaborative tools to developers. Consequently, the number of artifacts in the platform has grown significantly, making it a suitable data source for researchers to study software development from different perspectives. Parallel to this rise in popularity, there have been many advancements in terms of modelling graphical data. Notably, the problem of graph completion has been studied more thoroughly in the context of knowledge graphs. This thesis focuses on connecting these two disciplines.

Our study highlights the potential of adapting knowledge graphs to answer broad software engineering questions. Meanwhile, it also reveals the shortcomings of existing temporal knowledge graph embedding models to solve these questions. More precisely, we present a temporal knowledge graph based on the daily interactions between GitHub artifacts, which enables us to pose software engineering questions as queries over the knowledge graph. In particular, we first introduce three new datasets, each with distinguished properties, to study different aspects of GitHub. Then, we benchmark existing temporal knowledge graph embedding models on the newly introduced dataset, which reveals these models' unsatisfactory performance on extrapolated queries and time prediction queries. Therefore, we propose two novel extensions over existing temporal knowledge graph embedding models that drastically boost the newly introduced datasets' performance. Finally, we evaluate our model in a well-established scenario to study its applicability for solving the underlying software engineering question, which reveals a performance gap compared to our baselines; hence, we state our hypotheses to address this problem in future works.

Abrégé

Au cours des dernières années, GitHub est devenu la plateforme leader pour le développement de logiciels open-source, offrant de nombreux outils collaboratifs aux développeurs. Conséquemment, le nombre d'artefacts dans la plateforme a augmenté, cela est une source de données appropriée afin que les chercheurs étudient le développement de logiciels sous différents angles. Parallèlement à cette montée en popularité, il y a eu de nombreuses avancées en termes de modélisation des données graphiques, notamment, le problème de la complétion de graphes a été étudié de manière plus approfondie dans le contexte des graphes de connaissances. Ainsi cette thèse se concentre sur la connexion de ces deux disciplines.

Notre étude met en évidence le potentiel d'adapter les graphiques de connaissances pour répondre à des questions générales d'ingénierie logicielle. Parallèlement, il révèle également les lacunes des modèles d'intégration de graphes de connaissances temporelles existants pour résoudre ces questions. Plus précisément, nous présentons un graphe de connaissances temporelles basé sur les interactions quotidiennes entre artefacts dans GitHub, ce qui nous permet de poser des questions de génie logiciel sous forme de requêtes sur le graphe de connaissances. En particulier, nous introduisons d'abord trois nouveaux ensembles de données, chacun avec des propriétés distinctives pour étudier différents aspects de GitHub. Ensuite, nous comparons les modèles d'intégration de graphe de connaissances temporelles existants sur le jeu de données nouvellement introduit, ce qui révèle les performances insatisfaisantes de ces modèles sur les requêtes extrapolées et les requêtes de prédiction temporelle. Par conséquent, nous proposons deux nouvelles extensions sur les

modèles d'intégration de graphes de connaissances temporelles existants qui améliorent considérablement les performances sur les ensembles de données nouvellement introduits. Enfin, nous évaluons notre modèle dans un scénario bien établi pour étudier son applicabilité pour résoudre la question de génie logiciel sous-jacente, qui révèle un écart de performance par rapport à nos lignes de base; par conséquent, nous formulons nos hypothèses pour aborder ce problème dans les travaux futurs.

Acknowledgements

First and foremost, I would like to thank my supervisors, Prof. Jin Guo and Prof. Daniel Tarlow, who guided and supported me throughout my studies. I am incredibly grateful for all the time and effort they put into helping me shape my research skills and consistently providing feedback during this project and thesis.

I want to thank all of my friends and labmates whom I spent my days interacting with in the past two years. I also want to thank Sophie for her help with writing the French abstract of this thesis.

Finally, and most importantly, I thank my parents for their unconditional and continuing support and encouragement. I am extremely fortunate to have you in my life.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

GitHub[1] is the largest host of source code in the world. It provides a distributed version control service based on Git revision control system[2] containing more than 43 million public repositories, 163 million public issues, and one billion public commits [12]. Along with its main functionality, i.e., hosting source codes, it offers many essential collaborative features such as issue tracking system, task management tools, and code review infrastructure. These functionalities have made GitHub a unique large-scale collaborative social network [38].

Over the past decade, the available artefacts hosted on GitHub have proliferated. The platform tracks all the occurring events and interactions, and makes them accessible through a public API. Additionally, other cold storages such as GH Archive[3] provide historical dumps of these events. These factors have made GitHub one of the most important resources for researchers to study various procedural aspects of software development and open source ecosystems along with the behavioural characteristics of users and software engineers [7].

The questions of interest to SE researchers include but not limited to “When will an issue be closed?" [31, 45], “Will a pull request be merged and if yes when?" [16, 51], and “Who will review a pull request?" [63, 22].

[1] https://github.com/
[2] https://git-scm.com/
[3] https://www.gharchive.org/

Introduction

In the past few years, software engineering (SE) researchers have mainly focused on the analysis of repositories, issues, and other entities in an isolated environment using stationary features to carry out tasks such as classification and clustering. This focus on isolated artefacts has led to a lack of studies from a interdisciplinary perspective, e.g. network analysis, in the previous publications [7].

However, if we take a step back, there exists an immense temporal system of interacting events comprised of various artefacts present in the platform that dynamically affect and change each other. This colossal complex network of interacting entities provides an excellent opportunity for researchers to examine GitHub as a whole, e.g. how specific features change the dynamics of interaction in GitHub. Additionally, it potentially yields the discovery of behavioural and procedural patterns that could only be perceived from a temporal standpoint. Besides providing the opportunity to study GitHub as a single unit, leveraging this network is potentially advantageous in combination with existing models operating on stationary characteristics of artefacts. Additionally, there exists a potential opportunity to study the everyday tasks solely by observing the dynamics of the network to discern how far a model can go in learning patterns with almost no extra information from the underlying artefacts.

From the modelling outlook, graphs are the ideal mathematical constructs to represent this system of interacting artefacts. With the advent of graph neural networks (GNNs) [15, 47] and knowledge graph embeddings (KGEs) [6], recent years have witnessed a surge of interest in the field of graph representation learning. These models allow us to perform various tasks on graph structures, including node classification [32], relation prediction [6], and community detection [1]. However, the extent of tasks that could be carried out with these models is not limited to the tasks mentioned above and includes a wide variety of different tasks. Hamilton et al. [21] presents a comprehensive review of these models and their applications.

Introduced by Metaweb in 2007 [4], knowledge graphs (KGs) provide a robust structure for storing and querying the data. A KG represents entities and their relations with high reliability, explainability, and reusability. Notably, they depict the logical connections accurately and add logical induction capabilities to the model. Consequently, models could be built on top of these structures, enabling them to capture the interactions inside a sys-

tem better. In particular, when dealing with the temporal and dynamic aspects of the data compared to static characteristics, previous works have demonstrated [13, 2] that the temporal variations of knowledge graphs achieve better performance compared to the static variations.

This thesis is intended to inform future endeavours on the efforts made to adapt well-known software engineering questions to use the new paradigm of temporal graph representation learning, provide insight and analysis into the results, challenges, and shortcomings, and propose future research directions to build upon the work that has been done.

1.1 Contributions

The contributions of this work are:

1. Collecting three new temporal KG datasets from GitHub public events that allow casting the SE questions as time-conditioned prediction queries. Each dataset is particularly designed to allow the study of GitHub from a different perspective.
2. Implementing and benchmarking state-of-the-art knowledge graph completion methods on the newly introduced datasets.
3. Based on the observation that existing temporal Knowledge Graph Embeddings (KGEs) do not well capture patterns in relative time that are important to SE applications, particularly in extrapolated settings, e.g., “How long will it take from pull request being opened to being closed?," we propose a new relative temporal KGE, RT-X [2], inspired by the use of relative time in attention-based neural networks like Music Transformer [24] and Transformer-XL [8].
4. To further improve the proposed model, inspired by attention-based and relational GNNs such as Temporal Graph Networks [46] and R-GCN [48], we propose a novel relation-aware GNN architecture, i.e. R-TGN, that learns to attend over past events through the use of multi-head-attention, achieving state-of-the-art performance on the above tasks. The main advantage of this model is the replacement of the temporal context introduced in the previous model with a learning-based process.
5. Finally, we study the problem of predicting the pull request integrator when it is

opened and benchmark the proposed models and approaches.

1.2 Thesis Organization

The remaining of the thesis is organized as follows. Chapter 2 introduces the necessary machine learning background for the models used in this work, e.g. KGEs and GNNs. Chapter 3 presents an overview of previous works done on GitHub from a social perspective which essentially motivated this work. Chapter 4 states the efforts done on casting GitHub events as a KG and elaborates on the datasets, models, and results achieved in experiments. Chapter 5 provides insights into addressing shortcomings of the model in Chapter 4 and further expansion of our study on other parts of GitHub where we try out the generalizability of our best model in real-world scenarios. Chapter 6 sheds light on some possible future directions and presents the conclusion.

2

Background

This chapter introduces necessary machine learning background for the methods used in this work, starting with a brief introduction to classical machine learning models (Section 2.1), moving to matrix factorization models (Section 2.2), then introducing KGEs (Section 2.3), and finishing up with GNNs (Section 2.4).

2.1 Classical Machine Learning

As we are working with a new data source, training simple classical machine learning models as baselines would give us a benchmark on the collected datasets to be used for evaluating more complex models. We selected a few standard models for making classifications. The following establishes the strengths and weaknesses of these models.

Random Forest: Random Forest is an ensemble learner that makes use of the bootstrap aggregating process along with a voting mechanism to make predictions based on a fixed number of decision trees where each decision tree divides a continuous sample space by learning a set of axis-orthogonal hyperplanes. Although a decision tree can learn very sophisticated boundaries, it is sensitive to small perturbations in training data. However, this problem could be diminished by balancing the bias vs variance trade-off through the depth and number of decision trees used in a random forest.

Gaussian Naive Bayes: Naive Bayes is a simple probabilistic learner that assumes independence over features, making predictions as $\hat{y} = \text{argmax}_c \, p(c) \prod_i p(x_i|c)$. The Gaus-

sian variation, which is suitable to be used on continuous features, further assumes that each feature f_i is sampled from the underlying probability distribution $\mathcal{N}(\mu_i, \sigma_i^2)$ with μ_i and σ_i^2 respectively indicating the mean and the variance of the distribution. However, this assumption is usually too strong to hold on the feature space.

k-Nearest Neighbors (k-NN): k-NN is a non-parametric instance learner that makes use of a voting mechanism over the closest training samples on a specific metric space to make predictions. Although k-NN can learn highly irregular boundaries, it suffers from the curse of dimensionality, which causes all the points to be near each other in a high-dimensional feature space.

Note that all of these models are capable of outputing a probability distribution over the classes which we will use to obtain rankings over the candidates.

2.2 Matrix Factorization

In linear algebra, matrix decomposition is referred to as the act of factorizing a matrix into the product of two or more matrices. One example of such methods is LU-decomposition in which given a rectangular matrix A its decomposition is defined as

$$A = LU \tag{2.1}$$

where L and U are lower-triangular and upper-triangular matrices, respectively.

Inspired by the use of matrix decomposition in linear algebra, Shpak [50] introduced the first gradient-based matrix factorization algorithm. The goal of this algorithm is learning to generalize over the missing values of a sparse matrix K based on the non-empty elements. To this end, each entity is represented through an embedding function that maps elements in its domain to d-dimensional vector representations, e.g. $E : \mathcal{V} \rightarrow \mathbb{R}^d$, where d is a hyperparameter. Usually, an embedding is implemented as a lookup table to maintain efficiency. Then the regularized squared error objective is optimized using stochastic gradient descent as

$$\min_{q^*, p^*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{2.2}$$

where q_i and p_u are the vector representations of the two interacting entities, r_{ui} is a node-level statistic between them, and λ is the regularization coefficient. After optimizing Equation 2.2, the final approximation of matrix K would be achieved through a matrix decomposition defined as

$$K = PQ \, . \tag{2.3}$$

The aforementioned method relies on specific measurements between entities, e.g. rating given to a movie by a user; however, recent developments [43, 17] have seen the adaptation of stochastic measures. More specifically, the node2vec [17] algorithm employs a noise contrastive estimation approach [18] that approximates the optimization of cross-entropy loss over the whole sample space as

$$\mathcal{L} = \sum_{(u,i)\in\mathcal{K}} -\log \sigma(q_i^T p_u) + \gamma \mathbb{E}_{u_n \sim P_n(\mathcal{V})} \left[\log \sigma(q_i^T p_{u_n})\right] \tag{2.4}$$

where $P_n(\mathcal{V})$ denotes a noise distribution, usually a uniform distribution, over the sample space $\mathcal{V}$, $\gamma > 0$ is a hyperparameter balancing their weight of negative and positive samples, and σ is the sigmoid function.

2.2.1 Inference

Given a matrix factorization model and an entity q_i, the inference task is to find the entity with the highest similarity score defined as the dot product of the two entities, i.e.

$$\max_{u*} q_i^T p_u \, , \tag{2.5}$$

with the results being a sorted list of entities with regards to the calculated score.

2.3 Knowledge Graphs

Graphs are omnipresent structures with a powerful mathematical abstraction for representing and analyzing a wide variety of data allowing us to represent them as special-case graphs. For example, a sentence, or any sequence, could be represented as a path graph or an image could be represented as a lattice graph. Social networks are common real-world

examples of these structures.

A directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a collection of nodes $\mathcal{V}$ and a collection of edges

$$\mathcal{E} \subseteq \{(u, v) | (u, v) \in \mathcal{V} \times \mathcal{V}\} \, . \tag{2.6}$$

Hence, a suitable way to represent a graph is by an adjacency matrix $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ where

$$\mathbf{A}[u, v] = 1 \leftrightarrow (u, v) \in \mathcal{E} \tag{2.7}$$

To accommodate richer structures, we extend this definition to account for edge-types, e.g. follower or following relations in social networks, node-types, e.g. user or bot types in GitHub, and temporal relations, e.g. a user starts/ends following another user in social networks.

Multi-Relational Graphs: Adding a set of relation-types $\mathcal{R}$, a multi-relational graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ where

$$\mathcal{E} \subseteq \{(u, r, v) | (u, r, v) \in \mathcal{V} \times \mathcal{R} \times \mathcal{V}\} \, . \tag{2.8}$$

Each multi-relational graph is represented by an adjacency tensor $\mathcal{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{R}| \times |\mathcal{V}|}$ where

$$\mathcal{A}[u, r, v] = 1 \leftrightarrow (u, r, v) \in \mathcal{E} \, . \tag{2.9}$$

Heterogeneous Multi-Relational Graphs: Adding a set of node-types $\mathcal{C}$, a heterogeneous multi-relational graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{C}, f)$ where

$$f : \mathcal{V} \to \mathcal{C} \tag{2.10}$$

is a function mapping each node to its respective type.

Temporal Heterogeneous Multi-Relational Graphs: Adding a set of timestamps $\mathcal{T}$, a temporal heterogeneous multi-relational graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{C}, f, \mathcal{T})$ where

$$\mathcal{E} \subseteq \{(u, r, v, t) | (u, r, v, t) \in \mathcal{V} \times \mathcal{R} \times \mathcal{V} \times \mathcal{T}\} \, . \tag{2.11}$$

Each temporal heterogeneous multi-relational graph is represented by an adjacency tensor $\mathcal{A} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{R}|\times|\mathcal{V}|\times|\mathcal{T}|}$ where

$$\mathcal{A}[u,r,v,t] = 1 \leftrightarrow (u,r,v,t) \in \mathcal{E} \ . \tag{2.12}$$

To cover a broader range of structures, each graph could be assumed to be a multi-graph represented by an adjacency matrix/tensor with non-negative integer elements, i.e. whole numbers $\mathbb{W}$, instead of binary elements, i.e. $\{0,1\}$. This is a crucial requirement when we are dealing with models/tasks that require to have access to count of the same interaction.

Given the sets of entities $\mathcal{V}$, and relations $\mathcal{R}$, a static knowledge graph $\mathcal{G}_\mathcal{S}$ is the underlying heterogeneous multi-relational graph representing a subset of all true facts $\mathcal{W}_\mathcal{S} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$. Similarly, with the addition of a set of timestamps $\mathcal{T}$, a temporal knowledge graph $\mathcal{G}_\mathcal{T}$ is the underlying temporal heterogeneous multi-relational graph representing a subset of all true temporal facts $\mathcal{W}_\mathcal{T} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V} \times \mathcal{T}$. Additionally, relations in a knowledge graph (KG) could be bidirectional, represented by undirected edges.

In KGs, the *Open World Assumption* holds indicating that the non-existing edges are unknown. Hence, the problem of KG completion is the task of inferring the set of missing facts from the set of available facts, formally defined as

$$\mathcal{G} \models \mathcal{W} - \mathcal{G} \ . \tag{2.13}$$

2.3.1 Knowledge Graph Embeddings

A KGE aims to learn representations, usually a continuous vector, for entities, relations, and sometimes timestamps by leveraging the observed facts from a given KG. KGEs typically consist of an encoder generating representations for the inputs and a decoder generating a score based on those representations. Such models could be divided into static and temporal groups based on the kind of KG that they can represent.

The most common static KGE models either use translational transformations such as TransE [6] and RotatE [52] or bilinear transformations such as DistMult [62] and SimplE [28].

Model	Score Function	Symmetry	Antisymmetry	Inversion	Composition
SE [5]	$-\|W_{r,1}h - W_{r,2}t\|$	✗	✗	✗	✗
TransE [6]	$-\|h + r - t\|$	✗	✓	✓	✓
TransX [60, 39]	$-\|g_{r,1}(h) + r - g_{r,2}(t)\|$	✓	✓	✗	✗
DistMult [62]	$\langle h, r, t\rangle$	✓	✗	✗	✗
ComplEx [58]	$Re(\langle h, r, \bar{t}\rangle)$	✓	✓	✓	✗
RotatE [52]	$-\|h \odot r - t\|$	✓	✓	✓	✓

Table 2.1: Comparison of modeling abilities of common static KGEs [52]. TransX represents a wide range of models such as TransR [60] and TransH [39]

Expressivity: The inherent ability of a model to infer different relation patterns is referred to as expressivity. If the employed model is not expressive enough for the underlying KG, it will be likely to underfit/underperform on that specific KG. Symmetry, antisymmetry, inversion, and composition are among the most common relation patterns in KGs. Formally, these patterns are defined as

- **Symmetry:** r is symmetric if

$$\forall x, y : \ r(x, y) \Rightarrow r(y, x) \tag{2.14}$$

- **Anti-symmetry:** r is anti-symmetric if

$$\forall x, y : \ r(x, y) \Rightarrow \neg r(y, x) \tag{2.15}$$

- **Inversion:** r_1 is inverse of r_2 if

$$\forall x, y : \ r_2(x, y) \Rightarrow r_1(y, x) \tag{2.16}$$

- **Composition:** r_1 is composed of r_2 and r_3 if

$$\forall x, y, z : \ r_2(x, y) \wedge r_3(y, z) = r_1(x, z) \tag{2.17}$$

Table 2.1 presents a comparison of modeling abilities of common static KGEs. Given the illustrated comparison in Table 2.1, RotatE was chosen as the base model for the experi-

ments.

RotatE [52]: Built on top of two complex-valued embeddings $E_{\mathcal{V}}$ and $E_{\mathcal{R}}$ for entities and relations respectively, RotatE models each relation as a rotation between entities in the embedding vector space. The advantage of RotatE is its ability to infer all mentioned above relations patterns. Given a triplet (s, o, r), RotatE's scoring function is defined as

$$\phi(s, o, r) = -\|E_{\mathcal{V}}(s) \odot E_{\mathcal{R}}(r) - E_{\mathcal{V}}(o)\|. \tag{2.18}$$

With regards to temporal KGEs, methods for time prediction and time-conditioned link prediction in KGs [30, Section 5.1-5.2] are generally based on point process models [56, 57, 33] or adaptations of static KGEs that additionally use time to compute scores [9, 35, 11, 13]. While point processes models are elegant, they are more challenging to work with and require strong assumptions on their underlying intensity functions [see, e.g., 57, Equation 1 & Section 4]. Thus we focus on KG embedding-based methods, in particular starting with Diachronic Embeddings (DE-X) [13] for time-varying embeddings as it has demonstrated state-of-the-art performance over common benchmarks.

Diachronic Embeddings [13]: Diachronic embeddings are temporal extensions of embeddings defined as

$$D(e, t) = E_{\mathcal{V}}(e) \oplus (E_A(e) + \sin(t \times E_F(e) + E_\phi(e))) \tag{2.19}$$

where $\oplus$ is the concatenation operator and E, E_A, E_F, E_ϕ are embeddings and the last three respectively represent *Amplitude*, *Frequency*, and *Phase* of a multidimensional sinusoid.

Constructed on top of diachronic embeddings, DE-X defines a general extension to static models allowing them to operate on temporal KGs. The idea behind this extension is to replace embeddings in a static model with diachronic embeddings adding temporal information into the model without any changes to the scoring function.

The expressivity of any DE-X model is greater equal than the analogous static model as there exists a configuration for the respective DE model that is equivalent to the static variation. Moreover, DE-X models can infer entailment relation pattern, e.g. if someone becomes the president of a country then (s)he is a citizen of that country, defined as

- r_1 entails r_2 if

$$\forall x, y : \; r_2(x, y) \models r_1(x, y) \; . \tag{2.20}$$

Hence, to create a temporal variation of RotatE we replace its embeddings with diachronic embeddings to create DE-RotatE with a scoring function defined as

$$\phi(s, o, r, t) = -\|D(s, t) \odot E_{\mathcal{R}}(r) - D(o, t)\|. \tag{2.21}$$

2.3.2 Training

KGEs are trained using a noise contrastive estimation approach; however, since there are only positive samples in KGs, to avoid trivial solutions of embeddings a *Closed World Assumption* is made during the training phase indicating that non-existing edges are false. With this assumption, negative sampling is used to randomly select negative samples from a specific set, e.g. the whole sample space, with respect to a predefined distribution, e.g. uniform, to contrast with a given positive sample, optimizing [58]

$$\mathcal{L} = -\log \; \sigma(\gamma + \phi(x)) - \frac{1}{k} \sum_{i=1}^{k} \log \left(1 - \sigma(\gamma + \phi(x'_i))\right) \tag{2.22}$$

where x is the positive sample, $\{x'_i\}_{i=1,\dots,k}$ are the negative samples, and γ is a fixed margin acting as a regularizer.

Self-Adversarial Negative Sampling [52] is an extension of regular negative sampling scheme which replaces the uniform noise distribution with a distribution dependent on the current state of the embedding model defined as

$$P(x'_j | \{x_i\}_{i=1,\dots,k}) = \frac{\exp \alpha\phi(x'_j)}{\sum_i \exp \alpha\phi(x'_i)} \tag{2.23}$$

where α is the sampling temperature and $\{x_i\}_{i=1,\dots,k}$ is the set of negative samples. Due to the computational cost of realizing the actual distribution, these probabilities are embedded

into the objective function as

$$\mathcal{L} = -\log\ \sigma(\gamma + \phi(p)) - \sum_{i=1}^{k} p(n_i) \log\ (1 - \sigma(\gamma + \phi(n_i))) \tag{2.24}$$

Adapting negative sampling methods to temporal setting, Dasgupta et al. [9] demonstrate the potential of using a mixed negative sampling approach where negative samples are drawn from both Time Agnostic and Time Dependant negative samples distributions during training of a temporal KGE.

The last piece needed in the process of training KGEs is a regularization techinque to alleviate the chance of overfitting and improve generalization. Apart from common regularization techniques such as L1 and L2 regularizations [23, 54], Lacroix et al. [34] propose a novel regularization method based on nuclear norm-3 and demonstrate its efficiency for training KGEs.

2.3.3 Inference

A query on a given KG is represented as a tuple with missing elements denoting the desired search space to find the complete tuple with the highest score. Hence, each inference task on a KG is framed and executed as a proxy query. Moreover, the result of a query is a ranking over all the samples in the specified search space.

The set of all temporal queries is defined as

$$\mathcal{Q}_{\mathcal{T}} = \big\{(s, r, o, t) \mid s, o \in \mathcal{V} \cup \{*\}, r \in \mathcal{R} \cup \{*\}, t \in \mathcal{T} \cup \{*\}\big\} \tag{2.25}$$

with asterisk being a special symbol acting as the missing element to denote the desired search space. Since static KGs are special-cases of temporal KGs, we can easily derive the set of all static queries the same way as $\mathcal{Q}_{\mathcal{T}}$.

Link prediction is the most common type of query on KGs defined as inferring missing links given a KG and is represented as a tuple with exactly one entity missing, e.g. $(s, r, *)$ or $(*, r, o)$. Suggesting a connection in a social network is one of the widely used examples of these queries. From a propositional logic perspective, links are analogous to proposi-

tions and link prediction is analogous to one-step induction. In the temporal setting, link prediction queries are usually time-conditioned which means that they are conditioned on a specific given time, e.g $(*, r, o, t)$; however, it is also possible to execute time-independent link prediction queries such as $(s, r, *, *)$ or $(*, r, o, *)$.

Time prediction is another type of query that could be executed on temporal KGs. Predicting the time of a future connection in a social network is an example of this kind of query. These queries are represented as $(s, r, o, *)$.

2.4 Graph Neural Networks

Moving forward from encoders that encode relations and entities independently in KGEs, GNNs introduce a general framework to leverage the structure of the graph to generate representations on various levels. In comparison to previous well-known architectures, e.g. recurrent neural networks, the key distinguishing feature of GNNs is that they operate on an arbitrary ordered set of inputs which casts specific constraints on the functions that could be used to define a model. More specifically, given the adjacency matrix of input nodes A, the function f should satisfy either

$$f(PAP^T) = f(A) \quad \text{(Permutation Invariance) or} \tag{2.26}$$

$$f(PAP^T) = Pf(A) \quad \text{(Permutation Equivariance),} \tag{2.27}$$

where P is a permutation matrix. Another pivotal difference of GNNs to shallow embedding methods, e.g. KGEs, is their ability to use node-features when available.

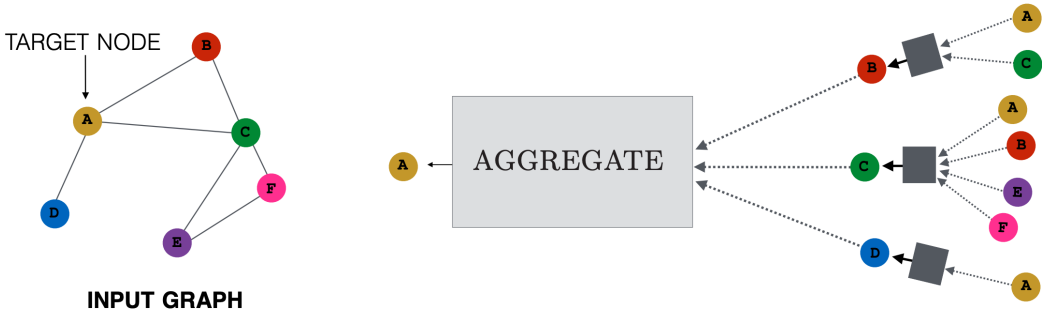

Figure 2.1: Overview of a 2-layer neural message passing algorithm on a single node [20].

2.4 Graph Neural Networks

The basic building block for GNNs is a form of neural message passing algorithm which relies on messages, i.e. continuous vector representations of neighbors, to pass on information among nodes and generate representation for nodes, subgraphs, and graphs. Battaglia et al. [3] define the generalized neural message passing algorithm at k-th layer of a GNN as

$$h_{(u,v)}^{(k)} = \text{UPDATE}_{edge}(h_{(u,v)}^{(k-1)}, h_u^{(k-1)}, h_v^{(k-1)}, h_{\mathcal{G}}^{(k-1)}) \tag{2.28}$$

$$m_{\mathcal{N}(u)} = \text{AGGREGATE}_{node}(\{h_{(u,v)}^{(k)}\}_{\forall v \in \mathcal{N}(u)}) \tag{2.29}$$

$$h_u^{(k)} = \text{UPDATE}_{node}(h_u^{(k-1)}, m_{\mathcal{N}(u)}, h_{\mathcal{G}}^{(k-1)}) \tag{2.30}$$

$$h_{\mathcal{G}}^{(k)} = \text{UPDATE}_{graph}(h_{\mathcal{G}}^{(k-1)}, \{h_u^{(k)}\}_{\forall u \in \mathcal{V}}, \{h_{(u,v)}^{(k)}\}_{\forall (u,v) \in \mathcal{E}}) \tag{2.31}$$

where $\mathcal{N}(u)$ is the neighborhood around the node u, $h_{(u,v)}$ is the edge representation, h_u is the node representation, and $h_{\mathcal{G}}$ is the graph representation. Figure 2.1 illustrates a simplified variation of a generalized 2-layer neural message passing algorithm. Complementary to the neural message passing algorithm, Hamilton et al. [19] introduce a simple neighbor sampling technique to keep the computational cost tractable by aggregating over a fixed-size random set of neighbors at each layer instead of the complete neighborhood.

Moving on to the temporal setting, Rossi et al. [46] introduce Temporal Graph Networks (TGN), a novel GNN architecture which is particularly designed to leverage the temporal nature of a given graph. Formally, TGN defines the representation generation process for node u at timestamp t, $h_u(t)$, at k-th layer as

$$h_u^{(k)}(t) = \text{MLP}^{(k)}(h_u^{k-1}(t) \parallel \widetilde{h}_u^{(k)}(t)), \tag{2.32}$$

$$\widetilde{h}_u^{(k)}(t) = \text{MultiHeadAttention}^{(k)}(q^{(k)}(t), K^{(k)}(t), V^{(k)}(t)), \tag{2.33}$$

$$q^{(k)}(t) = h_u^{k-1}(t) \parallel \phi(0), \tag{2.34}$$

$$K^{(k)}(t) = V^{(k)}(t) = C^{(k)}(t), \tag{2.35}$$

$$C^{(k)}(t) = \left[h_1^{(k-1)} \parallel e_1(t_1) \parallel \phi(t - t_1), \ldots, h_N^{(k-1)} \parallel e_N(t_N) \parallel \phi(t - t_N)\right] \tag{2.36}$$

where $\phi(.)$ is the time encoding presented in [29], $\parallel$ is the concatenation operator, the multi-head-attention is the scaled dot-product variation [59], and $\{e_n(t_n)\}_{n=1,\ldots,N}$ is the set of edge features over $\mathcal{N}(u)$. Here, $h_u^{(0)}$ could either be an embedding or a feature vector.

3

Related Work

Becoming the most popular social coding platform [7], GitHub provides ample opportunities for researchers to study it from different perspectives. This chapter reviews previous related research done on various aspects of GitHub with a focus on those using social elements. This work is motivated by the numerous information entanglements found across entities in GitHub.

3.1 Pull Requests and Issues

Given the pull-based development model of GitHub, Yu et al. [63] study the feasibility of relying only on social relations to recommend pull-request reviewers and compare it to traditional approaches. Their findings show that social network analysis approaches achieve similar performance as traditional approaches. Fused with traditional methods, they found significant improvements over both methods, indicating the importance of combining social factors with technical factors.

Similarly, Jiang et al. [25] focus on the problem of assigning core members for contribution evaluation. Utilizing follower and following relationships as a proxy of the social closeness between the contributor and the core member, they achieve superior performance in comparison to various baselines.

As resource allocation becomes more critical with the growth of a project, Kikas et al. [31] study the problem of issue lifetime prediction across 4000 projects through static,

dynamic, and contextual features. Their study highlights the importance of fusing dynamic and contextual features to build more robust predictive models. In an extension done by Rees-Jones et al. [45], authors found out that in some cases there are advantages to using cross-project models instead of local-learning methods too.

3.2 Repositories

Studying the forking process in GitHub, Jiang et al. [26] uncover similar behaviour to social networks such as Twitter and Reddit, suggesting that repository owners should utilize social relationships to promote and attract forks through the introduction of their repositories to popular developers with a high number of followers.

Closely related, Xu et al. [61] present a personalized software-recommendation system that leveraged the user's past social behaviours such as starring and forking. Their result shows that in combination with the content of each project, the proposed model outperforms the other two baselines.

Looking at the project recommendation problem from the onboarding process perspective, Liu et al. [40] make use of the social ties between existing project members and the new member. Their results indicate the importance of past social interactions, essentially confirming the preference of developers toward working with former frequent collaborators. Their study also shows that prior experience in the programming language of the project, number of commits in the project, and size of the team working on the project are predictors of a higher probability of success in the onboarding process.

3.3 Users

Working on the problem of finding influential developers, Liao et al. [37] propose a method based on influence propagation through heterogeneous network constructed on top of user behaviours. They achieve significant improvement over other link analysis algorithms while showing the importance of using commit information to outperform traditional techniques such as PageRank and HITS.

Focusing on finding the indicators of being accepted into a team, [41] try to find the type

and qualities of project interactions that affect joining an open-source software team both in isolation and combination. Their findings show the pull requests influence the decision the most with frequent discussions having a positive effect.

3.4 Ecosystem

Being critical to the survival of open-source projects, Qiu et al. [44] study the impact of social-networks formed by contributors on sustained participation. Their findings show a correlation associated between decreased risk of disengagement and contributing to projects with more familiar team members as indicated by having previous collaborations.

Studying community aspects of GitHub, Tamburri et al. [53] propose a tool for measuring community aspects characterising a software community and a community structure predictor for software systems. Their results indicate a reliable prediction of community structure patterns using software engineering data.

4

GitHub Events as a Knowledge Graph

This chapter focuses on the efforts made on casting GitHub events as KG, adapting some related SE tasks to queries, benchmarking exiting models, and finally description of new proposed models. The content of this chapter is based on a previous peer-reviewed work [2].

4.1 Dataset

4.1.1 Retrieval

To create the dataset, we retrieved from GitHub Archive all of the raw public events in GitHub in 2019. The knowledge graph was then constructed by tuples, each of which represents an individual event containing temporal information based on its type and a predefined set of extraction rules. The properties of the constructed KG is shown in the first row of Table 4.1, referred to as GITHUB-SE 1Y.

4.1.2 Extraction

As for the fact extraction strategy, Table 4.2 presents the set of extraction rules used to build the KG from raw events each representing a relation type. Although 80 extractions rules are defined in Table 4.2, the raw events that we used only contained 18 of them.

The codes presented in the Relation column of Table 4.2, when divided on underscore, are interpreted as *a)* the first and the last components respectively represent entity types of

Dataset	|V|	|E|	|R|	|T|	D_{MAX}	D_{MED}
GitHub-SE 1Y	125,455,982	249,375,075	19	365	3,519,105	2
GitHub-SE 1Y-Repo	133,335	285,788	18	365	73,345	1
GitHub-SE 1M	13,690,824	23,124,510	19	31	1,324,179	2
GitHub-SE 1M-Node	139,804	293,014	14	31	639	2
FB15k [6]	14,951	592,213	1,345	-	11,963	108
FB15k-237 [55]	14,541	310,116	237	-	8,642	38
WN18 [6]	40,943	151,442	18	-	1,040	6
WN18RR [10]	40,943	93,003	11	-	521	3
GitHub (Original) [57]	12,328	771,214	3	366	-	-
GitHub (Subnetwork) [33]	284	20,726	8	366	4,790	53.5
ICEWS14 [11]	7,128	90,730	230	365	6,083	3
ICEWS05-15 [11]	10,488	461,329	251	4017	52,890	5
YAGO15K [11]	15,403	138,056	34	198	6,611	5
Wikidata [35]	11,153	150,079	96	328	586	5
GDELT [36]	500	3,419,607	20	366	53,857	10,336

Table 4.1: Comparison of the newly created datasets to existing static and temporal KG datasets. Missing or unavailable statistics are represented as a dash.

the event's subject and object, *b) AO*, *CO*, *SE*, *SO*, and *HS* are abbreviations of extracted information from raw payloads[1] serving as distinguishers between different relation types among entities, and *b)* the second to the last component represents the concrete action taken that triggers the event.

4.1.3 Preprocessing

As evident from the statistics presented in Table 4.1, both the GitHub-SE 1Y and the GitHub-SE 1M have computationally intractable sizes. Therefore, we resort to further process the datasets into tractable size KGs that are feasible to experiment with by employing two distinct strategies.

The first strategy aims to retain maximum temporal information about particular SE projects. To achieve this, first, an induced sub-graph containing all related nodes was extracted for each node with type *Repository*. Then, for each sub-graph a popularity score

[1] https://developer.github.com/webhooks/event-payloads/

4.1 Dataset

Event Type	Head	Relation (Code)	Tail
Commit Comment	User	Actor (U_AO_CC)	Commit Comment
Fork	Repository	Fork (R_FO_R)	Repository
Issue Comment	User	Created (U_SO_C_IC)	Issue Comment
		Edited (U_SO_E_IC)	
		Deleted (U_SO_D_IC)	
	Issue Comment	Created (IC_AO_C_I)	Repository
		Edited (IC_AO_E_I)	
		Deleted (IC_AO_D_I)	
Issues	User	Opened (U_SE_O_I)	Issue
		Edited (U_SE_E_I)	
		Deleted (U_SE_D_I)	
		Pinned (U_SE_P_I)	
		Unpinned (U_SE_UP_I)	
		Closed (U_SE_C_I)	
		Reopened (U_SE_RO_I)	
		Assigned (U_SE_A_I)	
		Unassigned (U_SE_UA_I)	
		Locked (U_SE_LO_I)	
		Unlocked (U_SE_ULO_I)	
		Transferred (U_SE_T_I)	
	User	Assigned (U_AO_A_I)	Issue
		Unassigned (U_AO_UA_I)	
	Issue	Opened (I_AO_O_R)	Repository
		Edited (I_AO_E_R)	
		Deleted (I_AO_D_R)	
		Pinned (I_AO_P_R)	
		Unpinned (I_AO_UP_R)	
		Closed (I_AO_C_R)	
		Reopened (I_AO_RO_R)	
		Assigned (I_AO_A_R)	
		Unassigned (I_AO_UA_R)	
		Locked (I_AO_LO_R)	
		Unlocked (I_AO_ULO_R)	
		Transferred (I_AO_T_R)	
Member	User	Added (U_CO_A_R)	Repository
		Removed (U_CO_E_R)	
		Edited (U_CO_R_R)	
Pull Request Review Comment	User	Created (U_SO_C_PRC)	Pull Request Review Comment
		Edited (U_SO_E_PRC)	
		Deleted (U_SO_D_PRC)	
	Pull Request Review Comment	Created (PRC_AO_C_P)	Pull Request
		Edited (PRC_AO_E_P)	
		Deleted (PRC_AO_D_P)	
Pull Request Review	User	Submitted (U_SO_S_PR)	Pull Request Review
		Edited (U_SO_E_PR)	
		Dismissed (U_SO_D_PR)	
	Pull Request Review	Submitted (PR_AO_S_P)	Pull Request
		Edited (PR_AO_E_P)	
		Dismissed (PR_AO_D_P)	
Pull Request	User	Assigned (U_SO_A_P)	Pull Request
		Unassigned (U_SO_UA_P)	
		Review Requested (U_SO_RR_P)	
		Review Request Removed (U_SO_RRR_P)	
		Opened (U_SO_O_P)	
		Edited (U_SO_E_P)	
		Closed (U_SO_C_P)	
		Ready for Review (U_SO_RFR_P)	
		Locked (U_SO_L_P)	
		Unlocked (U_SO_UL_P)	
		Reopened (U_SO_R_P)	
		Synchronize (U_SO_S_P)	
	User	Assigned (U_AO_A_P)	Pull Request
		Unassigned (U_AO_U_P)	
	User	Review Requested (U_RRO_A_P)	Pull Request
		Review Request Removed (U_RRO_R_P)	
	Pull Request	Assigned (P_AO_A_R)	Repository
		Unassigned (P_AO_UA_R)	
		Review Requested (P_AO_RR_R)	
		Review Request Removed (P_AO_RRR_R)	
		Opened (P_AO_O_R)	
		Edited (P_AO_E_R)	
		Closed (P_AO_C_R)	
		Ready for Review (P_AO_RFR_R)	
		Locked (P_AO_L_R)	
		Unlocked (P_AO_UL_R)	
		Reopened (P_AO_R_R)	
		Synchronize (P_AO_S_R)	
Push	User	Sender (U_SO_C)	Repository
Star	User	Created (U_HS_A_R)	Repository
		Deleted (U_HS_R_R)	

Table 4.2: Extraction rules used to build the KG from raw events.

was calculated as $P(G) = W_1 \times S_G + W_2 \times T_G$ where S_G is the size of the graph, T_G is the time-span of the graph, and $W_1, W_2 \in \mathbb{R}^+$ are weight values. Finally, from the top three ranked repositories, we selected the *Visual Studio Code* repository to extract a one-year slice as it exercised more functionalities related to the target entities in this work, i.e. issues and pull requests. We name this dataset GITHUB-SE 1Y-REPO due to its repository-centric characteristics.

The second strategy aims at preserving the most informative nodes regardless of their type. We used a variation of Snowball Sampling [14] on all the events in December 2019. This sampled dataset, i.e. GITHUB-SE 1M-NODE, captures events across various projects and therefore, can be used to answer queries such as which project does a user start contributing at a certain time.

Algorithm 1 Snowball Sampling strategy used for extracting the GITHUB-SE 1M-NODE dataset.

Require: set of nodes V, set of edges E, sample size N, growth size S, initial sample size K
  $L \leftarrow sortDescending(V)$ w.r.t node degree
  $Q \leftarrow MaxPriorityQueue()$
  for $i = 1, ..., K$ **do**
    Q.put($L[i]$)
  end for
  $U \leftarrow Set()$
  while $size(U) < N$ **do**
    $V_u \leftarrow Q$.top() w.r.t node degree
    U.put(V_u)
    $U_s \leftarrow randomSample(E[V_u])$ with size S
    for $i = 1, ..., S$ **do**
      Q.put($U_s[i]$)
    end for
  end while

Algorithm 1 describes the snowball sampling technique used to create the GITHUB-SE 1M-NODE dataset. Focused on the breadth of the covered projects in a short span-of-time, it aims at preserving the most informative nodes regardless of their types. Algorithm 2 describes the temporal sampling approach used to create the GITHUB-SE 1Y-REPO dataset. Focused on the depth of a single project in a long span of time, it aims at preserving

maximum temporal information regarding particular repositories.

Algorithm 2 Temporal Sampling strategy used for extracting the GITHUB-SE 1Y-REPO dataset.

Require: set of nodes V, size importance factor W_1, time span importance factor W_2, sample size N

```
R ← extractRelated(V) {Repository node type only}
P ← Array(|R|)
for i = 1, ..., |R| do
    P_i ← calculatePopularity(R_i, W_1, W_2)
end for
S ← sorted(P)
U ← Set()
for i = 1, ..., |S| do
    if size(U) < N then
        U.union(S_i)
    end if
end for
```

4.1.4 Characteristics

In Table 4.1, we compare the variations of GITHUB-SE KG proposed in this work with commonly used datasets in the literature. Even the sampled down versions of our datasets are considerably larger in terms number of nodes. They have much higher edge to node ratios which translates into sparsity in graphs, but this sparsity level is close to what appears in GitHub as a whole. Additionally, similar to relations, each node in our datasets is also typed.

Trivedi et al. [57] also collects a temporal KG dataset from GitHub. However, this dataset is exclusively focused on the social aspects of GitHub, discarding repositories and only including user-user interactions, and it does not appear to be publicly available beyond raw data and a small subnetwork extracted in a follow-up work [33]. To differentiate our datasets, which focus on the SE aspects of GitHub, we append `-SE` to the dataset names.

The distinguishing characteristics of the proposed datasets, i.e. size, sparsity, node-typing, diversity, focus on SE aspects, and temporal nature, introduce a variety of engineering and theoretical challenges that make these datasets a suitable choice for exploring and

exposing the limitations of temporal knowledge graph embedding models.

4.2 Methodology

4.2.1 Baselines

We first examine the performance of the state-of-the-art KG embedding models on the GITHUB-SE 1M-NODE and GITHUB-SE 1Y-REPO datasets. We select RotatE [52] for the static settings considering its ability to infer *Symmetry*, *Antisymmetry*, *Inversion*, and *Composition* relational patterns. Moreover, we use DE-X [13] for the dynamic setting due to its superior performance on existing benchmarks and the fact that for any static model X there exists an equivalent DE-X model ensuring the ability to learn aforementioned patterns.

4.2.2 Proposed Approach

Motivation

Inspired by the use of relative temporal information in machine learning literature, and given the potential causal characteristic of SE events, we hypothesize that temporal KGEs could also benefit from such information provided as an extra signal.

The idea of using relative temporal information has been successfully employed in natural language processing [59, 8] and music generation [24]. These models formalize the intuition that temporal spacing between events is more central than the absolute time at which an event happened. We believe this framing is also appropriate for SE applications of temporal knowledge graphs: to predict if a pull request is closed at time t, it is more important to know how long it has been since the pull request was opened than it is to know t.

However, one of the challenges is that there are a lot of events, and we do not want to hard-code which durations are relevant. Instead, we would like the model to learn which temporal durations are important for scoring a temporal fact. As the number of related facts to an entity could be as high as a few thousand, we propose to pick a fixed-number of facts

as temporal context to provide as an input to the models.

The manually engineered temporal context tries to bake into the model the expert knowledge regarding which events are most likely to have an effect on the inference task at the query-time. The hypothesis is that the required temporal context could be collected from the recent facts involving the two entities being scored. Hence, we propose to take the most recent fact of each relation type for a given entity resulting in a fixed-size temporal context for each entity at the query-time.

Formalization (RT-X)

Let

$$\mathcal{H}(e, r, t) = \{t' \mid (s', r, o', t') \in \mathcal{G} \wedge (t' < t) \wedge (s' = e \vee o' = e)\} \tag{4.1}$$

be the set of times associated with facts involving entity e and relation r occurring before time t, and let

$$\delta(e, r, t) = t - \max \mathcal{H}(e, r, t) \tag{4.2}$$

be the relative time since a fact involving e and relation r has occurred. Hence, an entity's *relative temporal context* at query time t_q is

$$\Delta(e, t_q) = \begin{bmatrix} \delta(e, 1, t_q) \\ \vdots \\ \delta(e, |R|, t_q) \end{bmatrix} \in \mathbb{R}^{|R|} . \tag{4.3}$$

Moreover, to mimic the temporal gap of information occurring in the evaluation phase of extrapolated link prediction, we insert a random temporal information gap, t_q, during the training phase whenever the context is retrieved. Following the introduction of the aforementioned temporal context, we now turn attention to using the relative temporal context $\Delta(e, t)$ as an input to temporal KG embeddings. Our inspiration is the *Transformer* encoder, which has emerged as a successful substitute to more traditional Recurrent Neural Network approaches used for sequential [59, 8, 24] and structural [42] tasks. The core idea is to employ a variation of attention mechanism called *Self-Attention* that assigns importance scores to the elements of the same sequence.

4.2 Methodology

Unlike recurrence mechanism, the positional information is injected to the Transformer styled encoders by *1)* adding sine/cosine functions of different frequencies to the input [59, 8], or *2)* directly infusing relative distance information to attention computation in form of a matrix addition [49, 24]. Vaswani et al. [59] introduced a positional encoding scheme in form of sinusoidal vectors defined as $\rho(i) = [\rho_1(i), \ldots, \rho_d(i)]$, with

$$\rho_j(i) = \begin{cases} \sin(i/10000^{\frac{\lfloor j/2 \rfloor}{d}}) & j \text{ even} \\ \cos(i/10000^{\frac{\lfloor j/2 \rfloor}{d}}) & j \text{ odd} \end{cases}, \tag{4.4}$$

where i is the absolute position, and d is the embedding dimension. In the follow-up *Transformer-XL* model, Dai et al. [8] introduce a reparameterization of the relative attention where the attention score between a query element at position i and a key element at position j is defined as

$$\begin{aligned} A^{rel}_{i,j} = &\underbrace{E(i)^\top W_Q^\top W_{K,E} E(j)}_{(a)} + \underbrace{E(i)^\top W_Q^\top W_{K,\rho} \rho(i-j)}_{(b)} \\ &+ \underbrace{u^\top W_{K,E} E(j)}_{(c)} + \underbrace{v^\top W_{K,\rho} \rho(i-j)}_{(d)} \end{aligned} \tag{4.5}$$

where $E(i), E(j) \in \mathbb{R}^{d \times 1}$ are the i and j element embeddings, $u, v \in \mathbb{R}^{d \times 1}$, W_Q, $W_{K,E}$, $W_{K,\rho}$ are trainable $d \times d$ matrices, and $i - j$ is the relative position between i and j.

The main difference in our setting is that the above models compute a score based on a single relative time $i - j$, while our relative temporal context Δ contains $|R|$ relative times for each entity. Our approach is to score a tuple (s, r, o, t) based on the information available at query time t_q[2]. For each entity $e \in (s, o)$ we define a positional embeddings matrix P of relative times between t and the events in its relative temporal context $\Delta(e, t_q)$

[2]During training, t_q is the t of the positive sample, and during evaluation, t_q is set to the maximum timestamp in the training set.

as

$$P(e,t,t_q) = \begin{bmatrix} \rho(t - t_q + \delta(e,1,t_q)) \\ \rho(t - t_q + \delta(e,2,t_q)) \\ \vdots \\ \rho(t - t_q + \delta(e,|R|,t_q)) \end{bmatrix} \in \mathbb{R}^{|R| \times d}. \tag{4.6}$$

Intuitively, these relative times encode "If the event happened at time t, how long would it have been since the events in the relative time context?"

A learned, relation-specific row vector

$$W_P(r) \in \mathbb{R}^{1 \times |R|} \text{ for } r = 1, \ldots, |R| \tag{4.7}$$

chooses which rows of P are important, and then

$$\gamma(r,e,t,t_q) = W_P(r)P(e,t,t_q) \in \mathbb{R}^{1 \times d}\,, \tag{4.8}$$

abbreviated as $\gamma(r,e,t)$, embeds the relative temporal context of e, replacing $W_{K,\rho}\rho(i-j)$:

$$\begin{aligned} A_x^{rel} = &\underbrace{D(s,t)W(r)D(o,t)^\top}_{(a)} + \underbrace{\gamma(r,s,t)W_P\gamma(r,o,t)^\top}_{(d)} \\ &+ \underbrace{E(s)W_E\gamma(r,o,t)^\top}_{(b)} + \underbrace{\gamma(r,s,t)W_E^\top E(o)^\top}_{(c)} \end{aligned} \tag{4.9}$$

where $W(r)$ is a relation-specific weight matrix and W_E and W_P are tuple-agnostic weight matrices; however, this formulation is suitable for bilinear models. Hence, we derive a translational variation for the DE-RotatE model as

$$\begin{aligned} A_x^{rel} = &\underbrace{\|D(s,t) \circ E_R(r) - D(o,t)\|}_{(a)} \\ &+ \underbrace{\|E(s)W_E - \gamma(r,o,t)\|}_{(b)} + \underbrace{\|\gamma(r,s,t) - E(o)W_E\|}_{(c)} \end{aligned} \tag{4.10}$$

where $W(r)$ is replaced by an embedding lookup table $E_R(r)$. Intuitively, under this for-

mulation (a) capture entities compatibility and (b) and (c) capture entity-specific temporal context compatibility. In comparison, the existing models only include term (a) discarding terms (b) and (c).

Baking more expert knowledge into the process, we employed two re-ranking heuristics during the evaluation time for time-conditioned link prediction. First, each entity was only evaluated among entities with the same type. Next, we push down the ranks of entities with prior interactions with the given entity.

Complexity

Table 4.3 presents time and space complexity comparison between the existing models and the introduced RT-X model. Notice that, while yielding superior performance, the number of free-parameters introduced in our extension does not increase linearly with the number of entities which is one of the bottlenecks of training large KG embedding models. Table 4.4 presents the average runtime of each model for every 100 steps with batch size set to 64. All experiments were carried on servers with 16 CPU cores, 64GB of RAM, and a NVIDIA V100/P100 GPU.

Model	Computational Complexity	Free Parameters Complexity
RotatE	$O(d_s)$	$O(d_s(\|E\| + \|R\|))$
DE-RotatE	$O(d_s + d_t)$	$O((d_s + d_t)(\|E\| + \|R\|))$
RT-DE-RotatE (ours)	$O(d_s + d_t + d_s d_r + d_r \|R\|)$	$O((d_s + d_t)(\|E\| + \|R\|) + d_r^2 + d_s d_r)$

Table 4.3: Time and space complexity comparison of the models given static embedding dimension d_s, diachronic embedding dimension d_t, relative time embedding dimension d_r, entity set E, and relation set R.

Model	Number of Batches	Avg Training Runtime
RotatE	100	77s
DE-RotatE	100	80s
RT-DE-RotatE	100	87s

Table 4.4: Average runtime comparison of the models in seconds.

4.3 Evaluation and Results

4.3.1 Datasets

We use a 90%-5%-5% events split for constructing the train, validation, and test sets. For the interpolated queries the split was done randomly, whereas we split the data using event timestamps for the extrapolated queries. Table 4.5 presents details of the splits.

Dataset	Type	#Train	#Validation	#Test
GITHUB-SE 1M-NODE	Interpolated	285,953	3,530	3,531
	Extrapolated	281,056	2,104	3,276
	Standard Extrapolated	275,805	2,104	3,276
GITHUB-SE 1Y-REPO	Interpolated	282,597	1,595	1,595
	Extrapolated	269,789	2,281	1,472
	Standard Extrapolated	252,845	2,281	1,472

Table 4.5: Details of train, validation, and test splits.

4.3.2 Queries

For time-conditioned link prediction, we selected events related to the resolution of Github issues and pull requests due to their direct impact on software development and maintenance practice. Particularly, we used *"Who will close issue X at time T?"* and *'Who will close pull-request X at time T?"* for evaluation. For time prediction, we used the analogous time queries of the aforementioned queries for evaluation, i.e. *"When will issue X be closed by user Y?"* and *'When will pull-request X be closed by user Y?"*.

4.3.3 Metrics

We calculated the standard metrics to evaluate the model performance on the test set. Moreover, we use standard error to calculate confidence intervals and detect statistically indistinguishable results.

4.3.4 Hyperparameters

We tuned our models using the hyperparameter ranges reported in Table 4.6 for dropout, η, ω, and α resulting in total of 72 runs. Then, following the best hyperparameters achieved on RotatE and DE-RotatE models, we used dropout = 0.4, $\eta = 0.5$, $\omega = 6.0$, $\alpha = 3 \times 10^{-5}$, $\lambda = 5 \times 10^{-4}$, time-agnostic negative ratio = 256, time-dependant negative ratio = 32, batch size = 64, warm-up steps = 100000, warm-up α decay rate = 0.1, steps = 200000, and validation steps = 10000 for all experiments.

We apply L3 regularization parameterized by λ as introduced in Lacroix et al. [34] on E, E_A, W_E, and W_P.

Hyperparameter	**Range**
Dropout	$\{0.0, 0.2, 0.4\}$
η	$\{0.5, 1.0\}$
ω	$\{3.0, 6.0, 9.0\}$
α	$\{10^{-3}, 10^{-4}, 3 \times 10^{-5}, 10^{-5}\}$
λ	$\{10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$
d_s	$\{128, 96, 64, 32, 0\}$
d_a	$\{128, 96, 64, 32, 0\}$
d_r	$\{128, 64, 32, 0\}$

Table 4.6: Hyperparameter ranges used for experiments.

To make a fair comparison, we chose a base embedding size of 128 for all experiments. Subsequently, we only report on the combinations of static embedding dimension d_s values and diachronic embedding dimension d_t values presented in Table 4.6 where $d_s + d_t = 128$. We evenly distribute d_t among all diachronic embeddings to prevent giving models a distinct advantage in terms of free-parameters. As for the relative time embedding dimension d_r, we report on all the combinations in Table 4.6 with d_s and d_t respecting the stated restriction resulting in total of 17 experiments per dataset.

4.3.5 Results

For the extrapolated time-conditioned link prediction queries, after using the validation set for hyperparameter tuning, we retrained the selected models using both training and validation sets for evaluation. We also report the model performance without retraining in Table 4.8.

Dataset	Type	Model	HITS@1	HITS@3	HITS@10	MR	MRR
GITHUB-SE 1M-NODE	Interpolated	RotatE	47.58	76.66	88.95	**807.40**	0.6328
		DE-RotatE	47.98	76.92	88.87	**779.50**	0.6349
		RT-DE-RotatE (ours)	**49.70**	**78.67**	**90.48**	**773.90**	**0.6522**
	Extrapolated	RotatE	25.40	**49.02**	**57.54**	**4762.87**	0.3797
		DE-RotatE	**26.28**	48.53	**57.33**	**4840.16**	**0.3838**
		RT-DE-RotatE (ours)	**26.50**	**49.54**	**57.94**	**4891.81**	**0.3888**
GITHUB-SE 1Y-REPO	Interpolated	RotatE	44.05	57.14	**80.95**	18.54	0.5460
		DE-RotatE	42.17	53.88	76.88	24.67	0.5233
		RT-DE-RotatE (ours)	**48.93**	**60.96**	78.32	**14.47**	**0.5815**
	Extrapolated	RotatE	2.11	4.82	9.71	1917.03	0.0464
		DE-RotatE	1.77	4.08	9.10	1961.75	0.0402
		RT-DE-RotatE (ours)	**38.25**	**40.08**	**64.06**	**1195.02**	**0.4345**

Table 4.7: Performance comparison on time-conditioned Link Prediction. Results within the 95% confidence interval of the best are bolded.

In Table 4.7 we compare the model performance on the time-conditioned link prediction queries. On the GITHUB-SE 1M-NODE queries, our model slightly outperforms existing models in some cases, but the difference is statistically insignificant in others. On the GITHUB-SE 1Y-REPO, on the other hand, our RT-DE-ROTATE model shows a significant performance boost, particularly on the extrapolated time-conditioned link prediction queries, indicating the importance of using relative time as temporal context.

For the extrapolated time prediction queries on GITHUB-SE 1Y-REPO dataset, our model performed slightly better on HITS@1, HITS@3, and Mean Reciprocal Rank (MRR) than the other existing models while marginally surpassing the random baseline on all metrics. These results, detailed in Table 4.9, stress the necessity of having further studies on extrapolated time prediction queries.

4.3 Evaluation and Results

Dataset	Model	HITS@1	HITS@3	HITS@10	MR	MRR
GITHUB-SE 1M-NODE	RotatE	19.60	**38.37**	**45.54**	6437.30	0.2965
	DE-RotatE	20.97	**38.03**	**45.21**	6504.79	0.3005
	RT-DE-RotatE (ours)	**22.10**	**38.61**	**45.54**	**5782.83**	**0.3113**
GITHUB-SE 1Y-REPO	RotatE	0.41	1.49	2.45	2259.03	0.0141
	DE-RotatE	5.16	8.83	16.44	**1342.25**	0.0911
	RT-DE-RotatE (ours)	**38.59**	**40.01**	**43.27**	1613.70	**0.4034**

Table 4.8: Performance comparison on standard extrapolated time-conditioned Link Prediction. Results within the 95% confidence interval of the best are bolded.

Dataset	Model	HITS@1	HITS@3	HITS@10	MR	MRR
GITHUB-SE 1Y-REPO	RotatE	1.77	18.27	56.05	10.62	0.1675
	DE-RotatE	3.46	7.27	**60.73**	**9.35**	0.1724
	RT-DE-RotatE (ours)	**6.18**	**19.29**	55.91	9.40	**0.2073**
	Random	5.26	15.79	52.63	9.5	0.1867

Table 4.9: Performance comparison on extrapolated Time Prediction.

Figure 4.1 presents the importance matrix W_P between relations. As evident from non-symmetry of the matrix, the model learns query-relation conditioned importance scores.

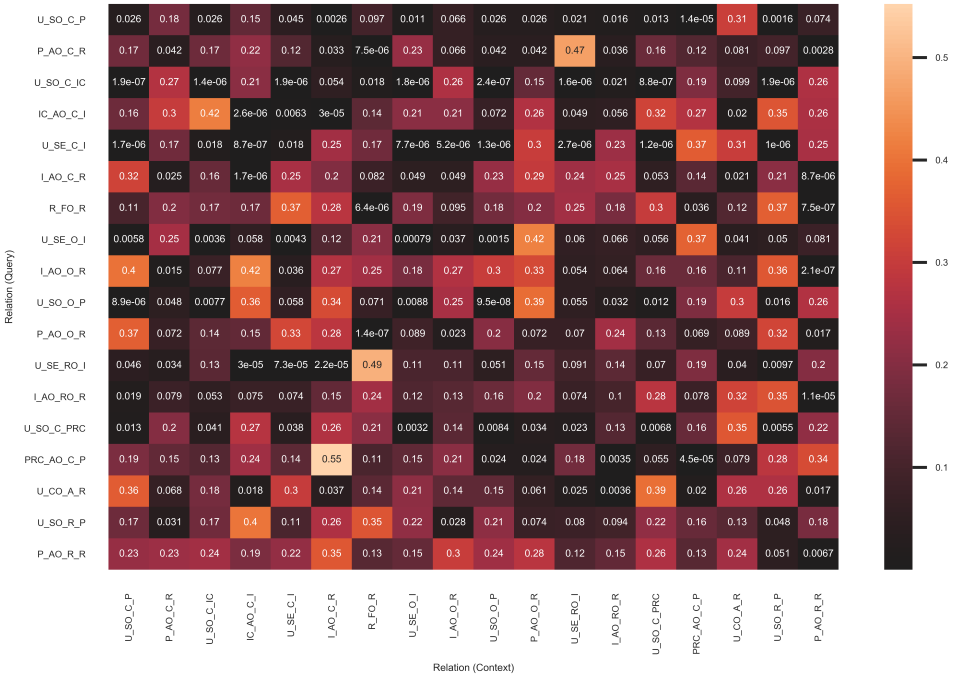

Figure 4.1: Example heatmap of the absolute importance scores among relations.

5

Real-World Applications

This chapter focuses on addressing the shortcomings of RT-X model as well as expanding the previous study to evaluate the generalizability of the proposed approaches in real-world scenarios.

5.1 Addressing Shortcomings

In Chapter 4 we observe that the main weakness of the RT-X model is its usage of a hand-picked temporal context which potentially limits its capabilities to infer temporal patterns. To alleviate this issue, we turn to the neural message passing algorithm and GNNs. We hypothesize that by giving the model the flexibility to gather information from past events autonomously, we could overcome the existing barrier and achieve improved performance.

5.1.1 Proposed Model (R-TGN)

Inspired by the state-of-the-art performance of TGN on various datasets, we propose a relation-aware extension, i.e. R-TGN, that takes advantage of temporal, relational, and contextual information available in the KG. The main difference of our model is its use of relation types to guide the information aggregation from past events. More specifically, we hypothesize that the information gathered from past events varies among different relations. Therefore, it would become crucial for the model to be aware of the relationship that is being evaluated. Formally, the node representation generation process of R-TGN given

a fact (s, r, o, t) is defined as

$$h_u(t) = \text{MLP}(E_{\mathcal{V}}(u) \parallel E_{\mathcal{R}}(r) \parallel \widetilde{h}_u(t)), \tag{5.1}$$

$$\widetilde{h}_u(t) = \text{MultiHeadAttention}(q(t), K(t), V(t)), \tag{5.2}$$

$$q(t) = E_{\mathcal{V}}(u) \parallel E_{\mathcal{R}}(r) \parallel \rho(0), \tag{5.3}$$

$$K(t) = V(t) = C(t), \tag{5.4}$$

$$C(t) = [E_{\mathcal{V}}(u_1) \parallel E_{\mathcal{R}}(r_1) \parallel \rho(t - t_1), \ldots, E_{\mathcal{V}}(u_N) \parallel E_{\mathcal{R}}(r_N) \parallel \rho(t - t_N)] \tag{5.5}$$

where $\rho(.)$ is the positional encoding proposed by Vaswani et al. [59], $\parallel$ is the concatenation operator, N is a hyperparameter controlling the number of past events to attend to, and

$$\{(s_n, r_n, o_n, t_n) | (s_n = u \vee o_n = u) \wedge t_n < t\}_{n=1,\ldots,N} \tag{5.6}$$

is the set of N most recent events before t during which u appears as either the source or target entity. The multi-head-attention operation used in TGN is a scaled dot-product attention mechanism [59]. In this model, we specifically use the 1-hop neighbourhood of each node; however, it is possible to extend the formalization to account for larger neighbourhoods.

Given the generated node representation, we then use the DistMult [62] scoring function to generate a score for the given fact. It is formally defined as

$$\phi(s, o, r, t) = \langle h_s(t), E_{\mathcal{R}}(r), h_o(t) \rangle = \sum_{i=1}^{d} h_s(t)[i] \times E_{\mathcal{R}}(r)[i] \times h_o(t)[i]. \tag{5.7}$$

5.1.2 Empirical Comparison between R-TGN and RT-X

Even though both RT-X and R-TGN are conceptually grounded in the idea that the occurrence of events is consequential to a series of prior events, they have different approaches to drawing and using this information from related events. RT-X relies on a handpicked set of events in combination with an auxiliary scoring function whereas R-TGN uses multi-head attention to merge all the contextual information into a single vector.

5.1 Addressing Shortcomings

We focus on extrapolated time-conditioned link prediction as the task for comparison using the GITHUB-SE 1Y-REPO dataset introduced in Chapter 4. This decision is based on the following reasons:

1. This dataset specifically resembles a more realistic scenario where we have all the information for a single project in a relatively long span of time rather than having the information for many projects in a short span of time.
2. In GitHub, we want to predict the person who closes an Issue or a Pull-Request in the future rather than in the past.

Starting our experiments with the simplest combination of components of R-TGN, we found out that even with those simple building blocks, it achieves improved performance over the best RT-X variation. We highlight the differences below:

1. Dropping the time-dependent negative-sampling.
2. Replacing the self-adversarial negative sampling loss with cross-entropy loss.
3. Dropping any regularization other that dropout used in multi-head-attention.
4. Only using regular embeddings rather than a fusion with diachronic embeddings.

Hyperparameter	**Range**
d_s	$\{32, 64, 128\}$
d_r	$\{32, 64, 128\}$
d_t	$\{32, 64, 128\}$
Time-agnostic Negative Ratio	$\{64, 128, 256\}$
N	$\{10, 20, 40\}$
Number of Attention Heads	$\{1, 2, 4\}$
Dropout	$\{0.0, 0.2, 0.4\}$
α	$\{10^{-3}, 3 \times 10^{-4}, 10^{-4}, 3 \times 10^{-5}\}$
Epochs	$\{50, 75, 100, 125\}$
t_q	$\{0, 7\}$

Table 5.1: Hyperparameter ranges used for tuning R-TGN.

5.1 Addressing Shortcomings

Hyperparameters

We tuned our model using the hyperparameter ranges reported in Table 5.1 by running 60 distinct random runs and taking the best set of hyperparameters based on the performance on the validation set. Following this procedure, the best set of hyperparameters found is d_s = 128, d_r = 32, d_t = 128, time-agnostic negative ratio = 256, N = 10, number of attention heads = 4, dropout = 0.2, $\alpha = 3 \times 10^{-4}$, epochs = 75, warm-up epochs = 37, warm-up α decay rate = 0.1, batch size = 64, t_q = 0, and validation epochs = 25. Note that in this setting d_t represents the positional encoding embedding dimension. Similar to our experiments on RT-X, to make a fair comparison, we chose a base embedding size of less than or equal 128 for all experiments.

Performance

Table 5.2 represents the performance comparison between the best RT-X model and the best R-TGN model. As evident from these results, we have gained significant performance boosts on all metrics, particularly 35.02% and 57.57% relative increases in **MRR** and **HITS@3** respectively. This shows the immense potential of building upon GNN based models that make use of temporal events for extrapolated time-conditioned link prediction.

Dataset	Type	Model	HITS@1	HITS@3	HITS@10	MR	MRR
GITHUB-SE 1Y-REPO	Extrapolated	RT-DE-RotatE	38.25	40.08	64.06	1195.02	0.4345
		R-TGN	**50.60**	**63.11**	**72.23**	**423.42**	**0.5867**

Table 5.2: Performance comparison on time-conditioned Link Prediction. Results within the 95% confidence interval of the best are bolded.

Runtime

Given the extra inherent computational complexity of using neural networks, we expect R-TGN to be much slower than RT-X model. However, as illustrated in Table 5.3, R-TGN has a significant lower runtime compare to RT-X. We believe that this is due to

1. Using regular embeddings in combination with non-learnable positional encodings instead of diachronic embeddings which reduces the gradient propagation computa-

tions significantly.

2. Attending to an optimal number of past events which could have a smaller size than the fixed-size handpicked temporal context, as it is the case for the best R-TGN model. This further emphasizes the importance of drawing information from the latest events involving a node to make more accurate decisions.

The experiments to test our hypotheses is left for future work.

Model	Number of Batches	Avg Training Runtime
RT-DE-RotatE	100	87s
R-TGN	100	12s

Table 5.3: Average runtime comparison of the models in seconds.

5.2 Study Expansion

In this section, we aim to expand our previous study to take into account certain constraints when deployed and used in practice. More specifically, we predict the integrator of a pull request when it is opened, i.e. we don't use any interaction with the pull request between its opening and closing times, whereas in previous tasks proposed in Chapter 4, the prediction is made when it is closed. Similar to GITHUB-SE 1Y-REPO, we use the task of extrapolated time-conditioned link prediction for pull requests as a proxy for the underlying task. Finally, we use an experimental scenario described in a previous work which will allow us to use well-established baselines as well as simple natural heuristics for performance comparison.

5.2.1 Dataset

Retrieval

To create the dataset, we retrieved from GitHub Archive all of the raw public events in GitHub from 2015 to 2019, i.e. five years worth of data. Then, we sort repositories based

on the number of pull requests that they had over the five years. Table 5.4 presents the statistics of the top-5 repositories based on this criteria.

Repository	Pull Requests
kubernetes/kubernetes	48,896
ansible/ansible	31,334
ceph/ceph	27,539
tgstation/tgstation	27,188
rust-lang/rust	25,018

Table 5.4: Top-5 repositories with the most number of pull requests from 2015 to 2019.

However, we drop *rust-lang/rust* from our study mainly because the repository uses a bot for closing all the pull requests automatically. Hence, we continue our experiments with the remaining 4 projects.

Extraction

As for the fact extraction strategy, Table 5.5 presents the set of extraction rules used to build the KG from raw events, each representing a relation type. In contrast to the 80 rules defined in Table 4.2, here we only use 14 distinct event types. The reasons for this reduction are *1)* dropping rules that don't appear in the raw data *2)* dropping rules that add a lot of low degree nodes, such as commits, that make the training intractable.

The codes presented in the relation column of Table 5.5, e.g. U_O_P, when divided on underscore, are *a)* the first and the last components respectively represent entity types of the event's subject and object, e.g. U for user and P for pull request, *b)* the second component represents the concrete action taken that triggered the event, e.g. O for opened.

Preprocessing

To validate our model and study its behaviour through time, we use a sliding window with a 3 months stride for the training time period. In other words, for $0 < N < 59$ we train on the first N month, validate on month $N + 1$, and finally test on month $N + 2$. Given the five years worth of data that we have, for each repository, we end up with 20 different datasets.

5.2 Study Expansion

Event Type	Head	Relation (Code)	Tail
Issues	User	Opened (U_O_I)	Issue
		Closed (U_C_I)	
		Reopened (U_RO_I)	
	Issue	Opened (I_O_R)	Repository
		Closed (I_C_R)	
		Reopened (I_RO_R)	
Pull Request Review Comment	User	Created (U_C_PRC)	Pull Request Review Comment
	Pull Request Review Comment	Created (PRC_C_P)	Pull Request
Pull Request	User	Opened (U_O_P)	Pull Request
		Closed (U_C_P)	
		Reopened (U_R_P)	
	Pull Request	Opened (P_O_R)	Repository
		Closed (P_C_R)	
		Reopened (P_R_R)	

Table 5.5: Extraction rules used to build the KG from raw events.

To process the training, validation, and test sets, we follow the procedure introduced by Jiang et al. [27]. Specifically, we take the following steps:

1. Filter out pull requests that have the same contributor, i.e. the person who has opened it, and integrator, i.e. the person who will close it.
2. Split the pull requests into training, validation, and test sets based on their opening times and use the opening time as the query time during inference.
3. Use all the valid integrators as the set of candidates for evaluation during the training period rather than considering the whole entity space.
4. We filter out pull requests in the validation and test sets whose integrators are not among the integrator set of the training set.

Comparing to the experiment design in other works on KG representation learning, this setting is much closer to the context of making predictions for real-world software engineering projects.

Finally, to enable our proposed model to draw some information from past interactions, we add a set of auxiliary facts to the evaluation sets which is only used as context of other nodes. More specifically, for each pull request, we add two additional facts, connecting it to its contributor and repository. These changes don't affect the real-world scenario as this information is available throughout the lifetime of a pull request.

5.2 Study Expansion

Category	Model	kubernetes/kubernetes	tgstation/tgstation	ceph/ceph	ansible/ansible
Bounds	UB	0.9725 ± 0.0072	0.9065 ± 0.0208	0.9549 ± 0.0162	0.9596 ± 0.0158
Interaction Based	MI	0.6494 ± 0.0266	0.3900 ± 0.0338	0.5373 ± 0.0434	0.3980 ± 0.0427
	MF	0.6855 ± 0.0245	0.3879 ± 0.0345	0.5056 ± 0.0427	0.4092 ± 0.0483
Temporal	MRI	0.6807 ± 0.0270	0.4249 ± 0.0337	0.5249 ± 0.0425	0.3994 ± 0.0429
	TMF	0.6975 ± 0.0220	0.3604 ± 0.0335	0.4523 ± 0.0410	0.3819 ± 0.0456
	STS	0.6649 ± 0.0280	0.4368 ± 0.0352	0.5541 ± 0.0445	0.3976 ± 0.0434
Fusion (GN)	TMF + STS + (RF + GN)	0.7478 ± 0.0204	$\mathbf{0.4777 \pm 0.0354}$	0.5516 ± 0.0422	0.4567 ± 0.0446
	MF + STS + (RF + GN)	0.7395 ± 0.0231	$\mathbf{0.4760 \pm 0.0356}$	$\mathbf{0.5863 \pm 0.0424}$	0.4623 ± 0.0457
	TMF + (RF + GN)	0.6672 ± 0.0231	0.3833 ± 0.0348	0.4950 ± 0.0412	0.4394 ± 0.0440
	MF + (RF + GN)	0.6763 ± 0.0243	0.4095 ± 0.0351	0.5618 ± 0.0422	0.4436 ± 0.0448
	MI + (RF + GN)	0.7751 ± 0.0195	$\mathbf{0.4788 \pm 0.0354}$	$\mathbf{0.5982 \pm 0.0421}$	$\mathbf{0.5117 \pm 0.0459}$
	MRI + (RF + GN)	0.7870 ± 0.0176	$\mathbf{0.4922 \pm 0.0356}$	0.5606 ± 0.0420	$\mathbf{0.4923 \pm 0.0450}$
	MI + MRI + (RF + GN)	0.7906 ± 0.0189	$\mathbf{0.4881 \pm 0.0356}$	$\mathbf{0.5930 \pm 0.0425}$	$\mathbf{0.5098 \pm 0.0458}$
	STS + (RF + GN)	$\mathbf{0.8151 \pm 0.0172}$	$\mathbf{0.4851 \pm 0.0357}$	$\mathbf{0.5681 \pm 0.0420}$	$\mathbf{0.4934 \pm 0.0448}$
Fusion (k-NN)	TMF + STS + (RF + k-NN)	0.7535 ± 0.0201	$\mathbf{0.4775 \pm 0.0357}$	0.5559 ± 0.0421	$\mathbf{0.4748 \pm 0.0454}$
	MF + STS + (RF + k-NN)	0.7567 ± 0.0220	$\mathbf{0.4705 \pm 0.0355}$	$\mathbf{0.5949 \pm 0.0424}$	$\mathbf{0.4736 \pm 0.0454}$
	TMF + (RF + k-NN)	0.6913 ± 0.0234	0.3946 ± 0.0354	0.5090 ± 0.0415	0.4585 ± 0.0440
	MF + (RF + k-NN)	0.6785 ± 0.0245	0.4084 ± 0.0347	0.5646 ± 0.0420	0.4690 ± 0.0450
	MI + (RF + k-NN)	0.7872 ± 0.0196	$\mathbf{0.4857 \pm 0.0355}$	$\mathbf{0.6066 \pm 0.0421}$	$\mathbf{0.5169 \pm 0.0455}$
	MRI + (RF + k-NN)	0.7845 ± 0.0194	$\mathbf{0.4819 \pm 0.0355}$	0.5581 ± 0.0415	$\mathbf{0.4981 \pm 0.0449}$
	MI + MRI + (RF + k-NN)	0.7869 ± 0.0194	$\mathbf{0.4897 \pm 0.0357}$	$\mathbf{0.6071 \pm 0.0421}$	$\mathbf{0.5114 \pm 0.0454}$
	STS + (RF + k-NN)	0.7804 ± 0.0195	$\mathbf{0.4869 \pm 0.0359}$	$\mathbf{0.5821 \pm 0.0423}$	$\mathbf{0.4999 \pm 0.0450}$
Bounds	R	0.0617	0.1694	0.0961	0.1658

Table 5.6: Performance comparison of all baselines. Results within the 95% confidence interval of the best are bolded.

Characteristics

Figure 5.1 and Figure 5.2 provide insights into the characteristics of the datasets. As evident, all repositories have a linear growth of the number of pull requests while having an extremely small node to edge ratio.

5.2.2 Baselines

We divide our baselines into three different categories, interaction based, temporal, and fusion, and then choose the best performing baseline from each category for the final comparison with the proposed model. All of the baselines introduced here only use information available in pull requests.

5.2 Study Expansion

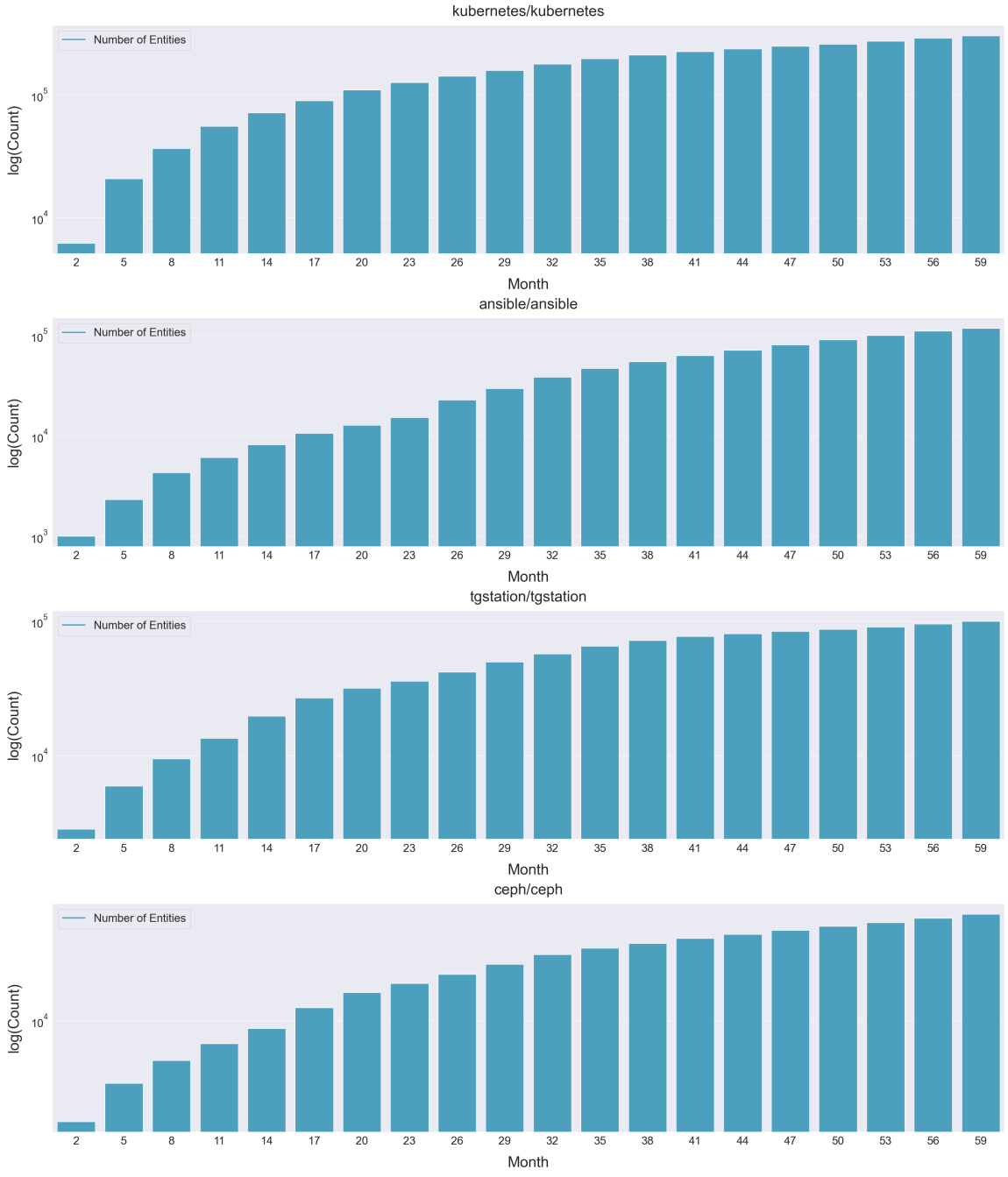

Figure 5.1: Entity counts, in log scale, from all sliding windows datasets on selected repositories.

5.2 Study Expansion

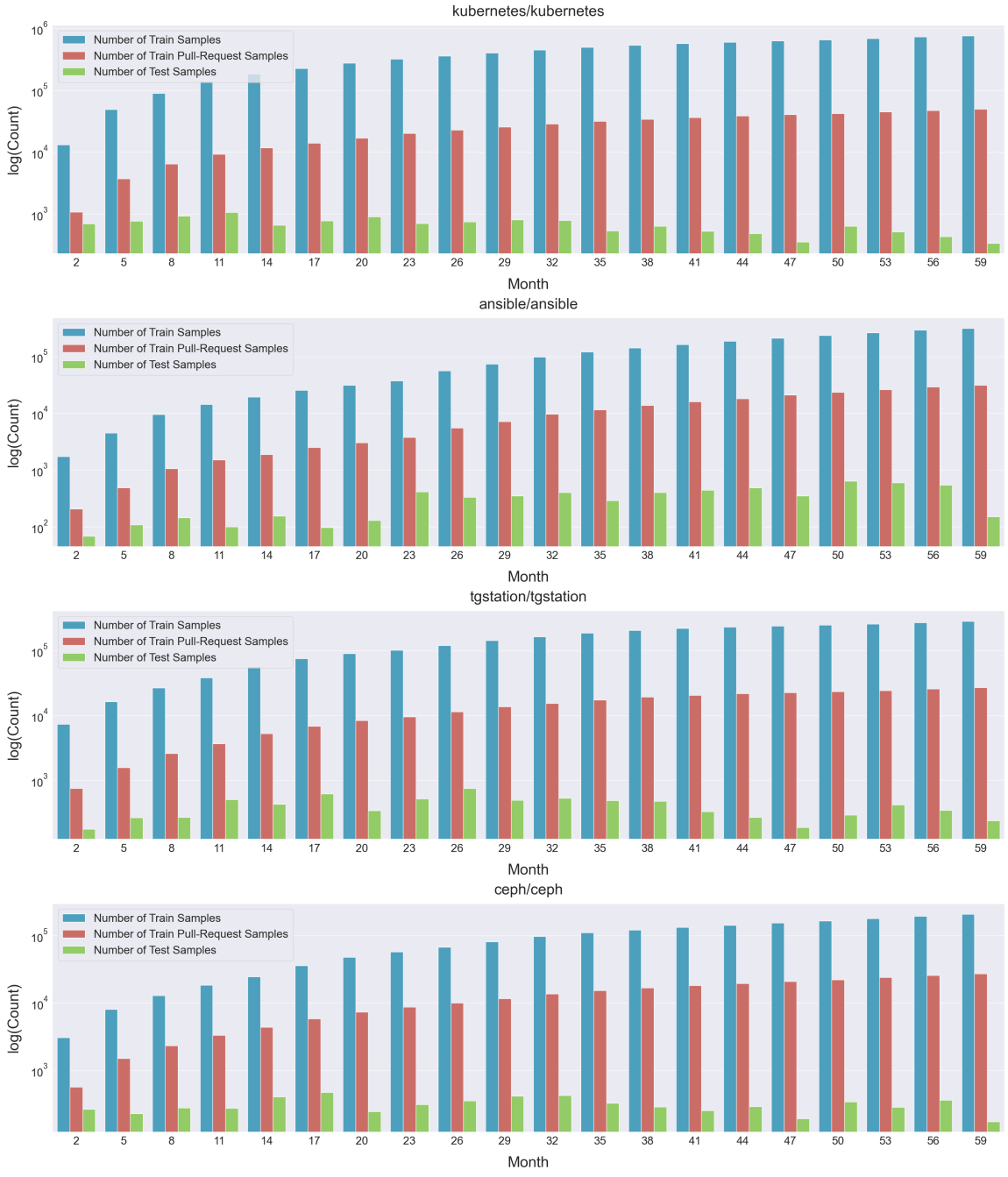

Figure 5.2: Split statistics, in log scale, from all sliding windows dataset on selected repositories.

Interaction Based

We use two interaction based baselines for comparison. The simplest baseline in this category is sorting the candidates based on the number of past interactions. The second baseline is a matrix factorization method using regular embeddings between integrators and contributors. Figure 5.3 illustrates the performance comparison of the above mentioned models.

As evident from Figure 5.3, the matrix factorization method almost always performs as well or better compared to the most interactions approach. This is particularly interesting given the fact that intuitively we expect it to perform as well at best. We believe this could be explained by the fact that the representation of each integrator and contributor in the d-dimensional space is affected by other entities which prevent model to overfitting; hence, allowing it to have more generalizability compared to simple heuristics. Table 5.6 presents the average performance of each model over all months.

Temporal

We use three temporal based baselines for comparison. The simplest baseline in this category is sorting the candidates based on the least elapsed time since the last interaction. Moreover, the second baseline is a temporal matrix factorization method using diachronic embeddings between integrators and contributors.

Furthermore, we also use the sorted time-decaying relationship score introduced by Jiang et al. [27] as our third baseline. Formally, this score is defined as

$$s(p_n, i) = \sum_{p_j \in C_n \cap I_i} (t_n - t_j)^{-1} \tag{5.8}$$

where p_n is the pull request, C_n is the set of pull requests opened by the contributor of p_n, I_i is the set of pull requests closed by integrator i, t_n and t_j are the opening times of the pull requests p_n and p_j respectively. Figure 5.4 presents the performance comparison of the above mentioned models

In the case of temporal baselines, we don't have a clear winner model over all datasets. Therefore, we choose the best model for each repository based on the average over all months as presented in Table 5.6.

5.2 Study Expansion

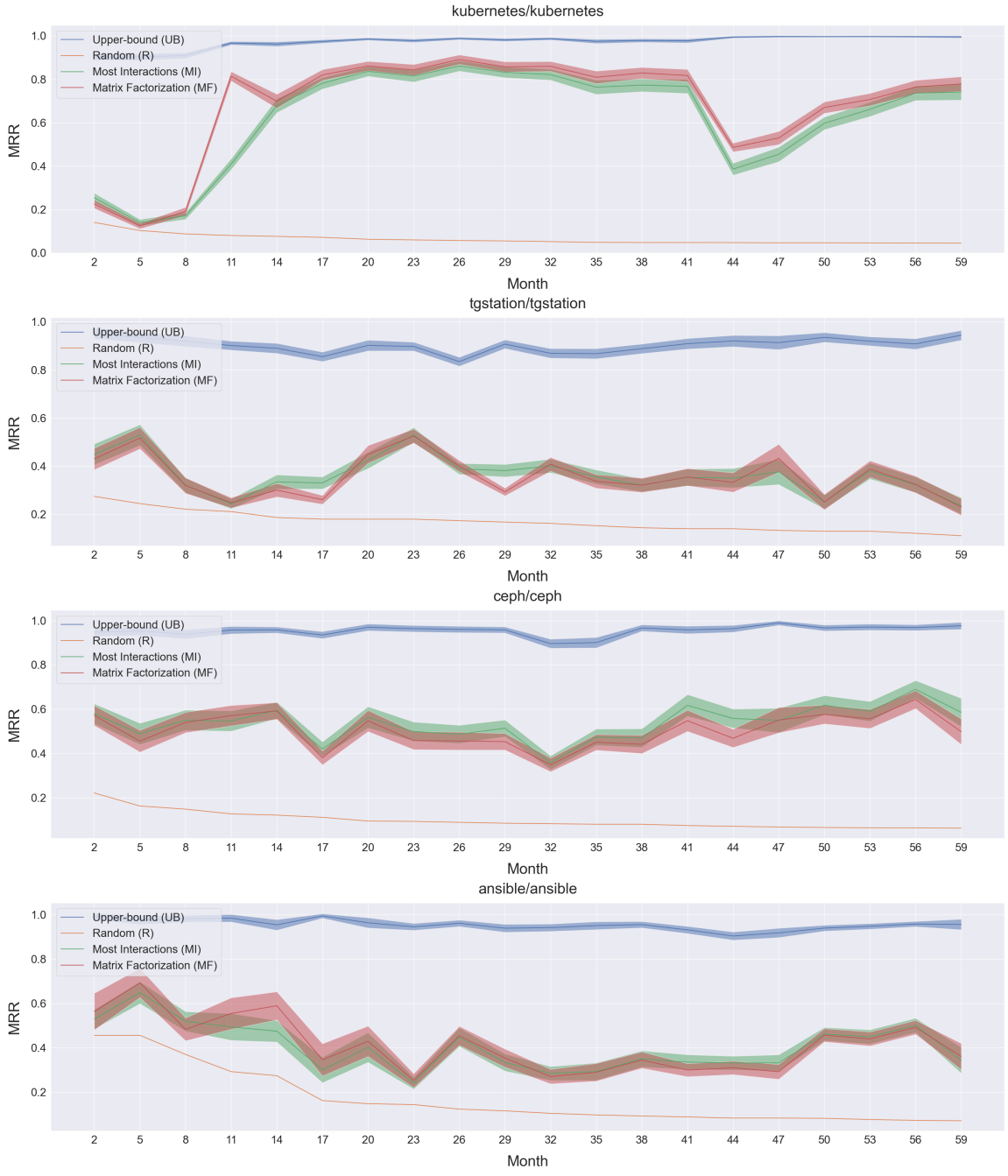

Figure 5.3: Interaction based baselines performance over all datasets. The area around each line indicates its error bars.

5.2 Study Expansion

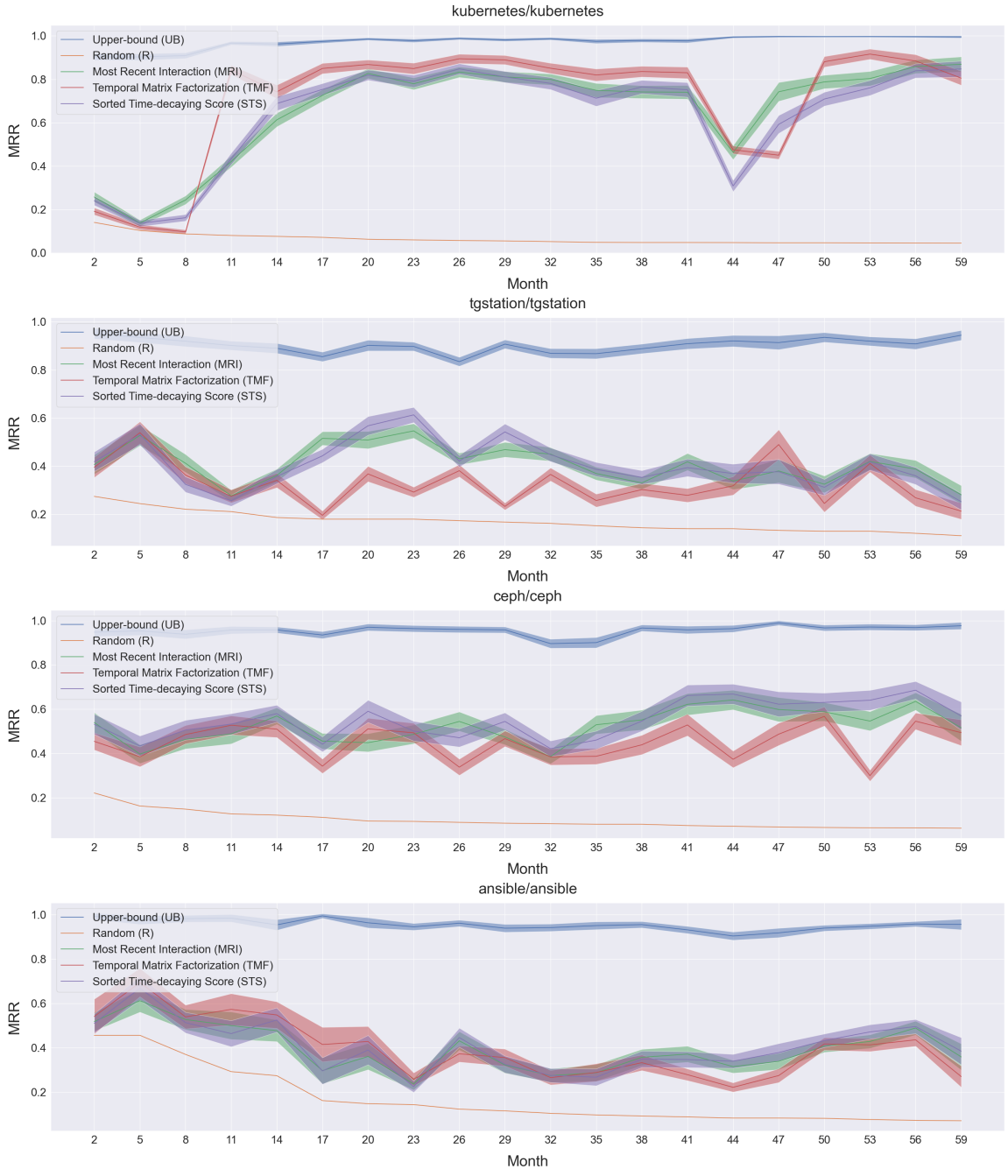

Figure 5.4: Temporal baselines performance over all datasets. The area around each line indicates its error bars.

Fusion

Inspired by the mixture model used by Jiang et al. [27], we stack all of our previous feature-sets with a random forest and a gaussian naive bayes as follows:

$$P(i|p_n) = \gamma P_{RF}(i|p_n) + (1-\gamma)P_{GN}(i|p_n) \tag{5.9}$$

where P_{RF} represents the probability obtained from the random forest and P_{GN} represents the probability obtained from the gaussian naive bayes. We also experiment with replacing the gaussian naive bayes with a k-NN. To stack with the classifiers properly, we normalize our features applying the followings:

1. For the number of interactions, we normalize it into a distribution over the set of candidates.
2. For the elapsed time from the last interaction, we use the inverse of this value.

As we don't have a clear winner model over all datasets, we choose the best model of each category based on the average over all months as presented in Table 5.6. Appendix A provides more insights on the performance of fusion models.

An interesting phenomena that we observed during our experiments, was a sudden change in performance on *kubernetes/kubernetes* at month 44. After investigating the external reasons, we found out that the repository has switched to a bot-based pull request closing system at that month. This is particularly interesting as it tests the ability of models to adapt to sudden distributional changes which is something that could happen in real-world applications. We particularly attribute the superior performance of *STS + (RF + GN)* on *kubernetes/kubernetes* to this quick adaptation. This could be observed both in Figure A.1 and Table 5.6.

To tune our stacking models we use a grid search over the provided ranges in Table 5.7 for each dataset. We also report the upper-bound and random baselines. It is essential to note that the small difference between the upper-bound and the perfect prediction is due to situations where the same contributor has pull requests that are opened at the same day but are closed by different integrators. In these situations model is forced to make a ranking on

those integrators which will always result in a subpar performance.

Model	Hyperparameter	Range
Random Forest	Number of Estimators	$\{15, 35, 55, 75, 95\}$
	Max Depth	$\{3, 5, 7, 9, 11\}$
Gaussian Naive Bayes	Variable Smoothing	$\{10^{-9}, 10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}\}$
k-NN	Number of Neighbors	$\{1, 25, 75, 125, 175, 225\}$
	Weights	$\{\text{uniform}, \text{distance}\}$
-	γ	$\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

Table 5.7: Hyperparameter ranges used for tuning stacking model.

To illustrate the distribution shifts present in our datasets we calculate the inverted average cosine distance between the contributor-integrator distribution in the training sets and validations sets. Formally, for an arbitrary dataset with N distinct contributors and M distinct integrators in the test set, this metric defined as

$$A_i = \left[P_{\text{train}}(I_1|C_i), P_{\text{train}}(I_2|C_i), \ldots, P_{\text{train}}(I_M|C_i)\right]^T \tag{5.10}$$

$$B_i = \left[P_{\text{test}}(I_1|C_i), P_{\text{test}}(I_2|C_i), \ldots, P_{\text{test}}(I_M|C_i)\right]^T \tag{5.11}$$

$$\text{AVG}_{\text{cos}} = 1 - \frac{1}{N}\sum_{i=1}^{N} \frac{A_i.B_i}{\parallel A_i \parallel \parallel B_i \parallel} \tag{5.12}$$

where I_j is the j-th integrator, C_i is the i-th contributor, and $P(I_j|C_i)$ is the conditional probability of I_j being the integrator given C_i as the contributor. Figure 5.5 presents the results of this metric over our evaluation datasets. When compared to the performance of baselines, we can see that they roughly follow the same trend, which is particularly interesting as we are more interested in models that could generalize despite these shifts; however, from the experiments, we could see that drastic distributional changes still pose a problem to these models. Further investigations for understanding these models' behaviors and sensitivities to input distribution shifts are left to future works. Another takeaway from this figure is that it illustrates the limitations of solely using pull requests on the model's abilities, which further reinforces the idea of using other entities to build better models.

5.2 Study Expansion

Figure 5.5: Inverted average cosine distance between training set predictions and test set predictions.

5.2.3 Proposed Approach Results

We tuned our model using the hyperparameter ranges reported in Table 5.1 by running 20 distinct random runs and taking the best set of hyperparameters based on the performance on the validation set. To keep the computation tractable across all windows and repositories, we tuned our model on 12 months windows and then used those hyperparameters for all preceding windows. In total we tuned our model on 20 settings, running a total of 400 experiments.

Hyperparameter	Range
d_s	$\{32, 64, 128\}$
d_r	$\{32, 64, 128\}$
d_t	$\{32, 64, 128\}$
Time-agnostic Negative Ratio	$\{128, 256\}$
N	$\{10, 20, 30\}$
Number of Attention Heads	$\{2, 4\}$
Dropout	$\{0.2, 0.4\}$
α	$\{10^{-3}, 3 \times 10^{-4}, 10^{-4}\}$
Epochs	$\{50, 75, 100\}$
t_q	$\{0, 7\}$

Table 5.8: Hyperparameter ranges used for tuning R-TGN.

Figure 5.6 illustrates the comparison between R-TGN and best performing baselines of each category. As evident, the model is performing inconsistently over different months with a lot of performance fluctuation. Given the facts that *1)* for every TMF or MF model there exists an equal R-TGN model and *2)* TMF and MF models are performing better on many datasets compared to their counterpart R-TGN models, we believe that one of the main reasons of our model underperforming is the tuning scheme that we used for finding optimal hyperparameters. Given the massive computational cost of tuning the model for every dataset, a more rigorous hyperparameters tuning is left for future work.

Despite the overall poor performance of our model, on *tgstation/tgstation* dataset it had an on-par performance on few occasions compared to more sophisticated baselines which have access to all of the scores before during the inference phase. Moreover, our

5.2 Study Expansion

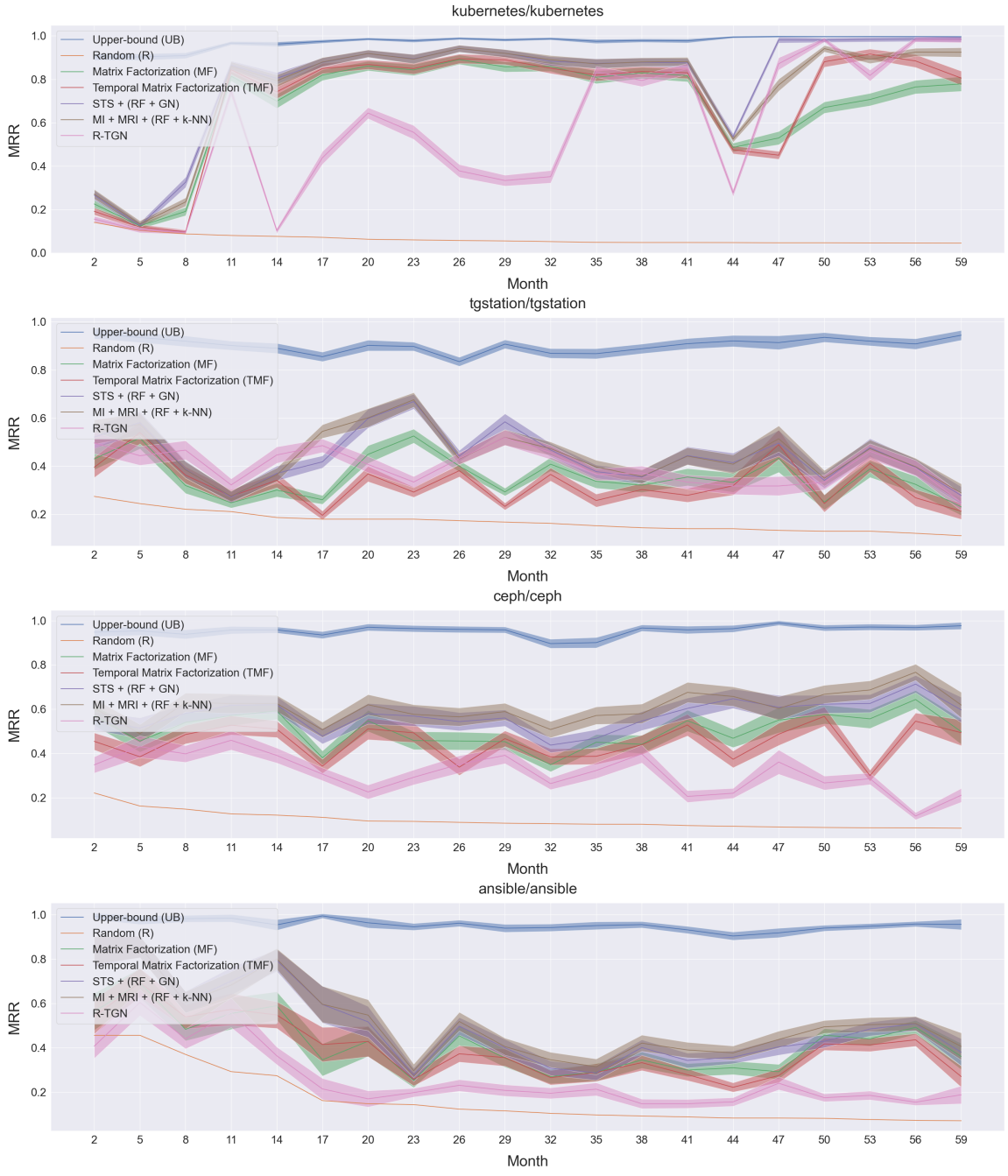

Figure 5.6: R-TGN performance comparison to best performing baselines of each category. The area around each line indicates its error bars.

model had a relatively fast recovery compared to baselines after the paradigm shift on *kubernetes/kubernetes* dataset at month 44. However, given the exceptional occurrences of these phenomena, further investigation in future works is needed to determine whether this could be replicated consistently.

To alleviate the shortcomings of R-TGN, we present two hypotheses to be tested in future works. First, we believe that we have to push back the time of all events related to a pull request or an issue, e.g., comments, to the opening time in order to mimic the evaluation settings. By doing so, the model would learn to make decisions in an information deprived situation that closely resembles the evaluation setting. Second, during the training and validation phases, we observed that the validation results had fluctuations between consecutive epochs, indicating a potential problem with the used training scheme. We believe that these issues could be mitigated by employing other regularization methods in combination with dropout. Further experimenting for testing these hypotheses is left to future work.

6

Future Work and Conclusion

6.1 Future Work

Throughout this thesis, we focused on the idea of solely using the dynamics present in the GitHub to make our inferences; however, each entity in our KG could be represented by features extracted from the underlying entities and their properties, e.g. the title of a pull request or an the textual content of an issue comment. We believe enriching the nodes with these features could potentially lead to a significant boost in performance. Furthermore, it would be interesting to see how we can combine various modalities, i.e. natural language and code, to enhance our existing models. The ultimate goal would be to devise models that can coherently represent the contents and properties of the entities and their interactions in GitHub. We believe that the paradigm of temporal graph representation learning provides promising tools to achieve this goal in the future.

6.2 Conclusion

In this thesis, we bridged between the SE domain questions and the literature on KGEs by introducing three new datasets based on the daily interactions in the GitHub platform and casting those questions as queries on an appropriate KG. Furthermore, we introduced RT-X and R-TGN, two novel extensions to existing KGEs that make use of past relevant events during inference time. Our initial experiments highlighted some of the shortcomings of existing temporal KGEs, notably on extrapolated time-conditioned link prediction,

and exhibited the advantage of leveraging past events as used in both RT-X and R-TGN models. However, despite these improvements on existing KGEs, our study expansion on a scenario taken from previous works revealed a performance gap between well-established baselines and our best model, i.e., R-TGN. Additionally, we presented a brief discussion on the potential reasons for this performance deficiency along with possible future directions for further investigations.

In total, this work highlighted new opportunities for improving temporal KGEs on time-conditioned link prediction queries and some of the shortcomings of the proposed models on existing evaluation scenarios. It also revealed the need for having well-thought-out experiment scenarios, including but not limited to careful consideration of practical constraints and selecting well-established baselines, when discussing the performance of complex models while paving the way for employing graph representation learning methods to answer software engineering questions.

Bibliography

[1] Monica Agrawal, Marinka Zitnik, Jure Leskovec, et al. Large-scale analysis of disease pathways in the human interactome. In *PSB*, pages 111–122. World Scientific, 2018.

[2] Kian Ahrabian, Daniel Tarlow, Hehuimin Cheng, and Jin LC Guo. Software engineering event modeling using relative time in temporal knowledge graphs. *arXiv preprint arXiv:2007.01231*, 2020.

[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[5] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Conference on artificial intelligence*, 2011.

[6] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[7] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. A systematic mapping study of software development with github. *IEEE Access*, 5:7173–7192, 2017.

[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[9] Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2001–2011, 2018.

[10] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] Alberto García-Durán, Sebastijan Dumančić, and Mathias Niepert. Learning sequence encoders for temporal knowledge graph completion. *arXiv preprint arXiv:1809.03202*, 2018.

[12] GitHub. Github public entities, 2020. URL https://github.com/search?q=is%3Apublic&type=Repositories. [Accessed: 2020-08-09].

[13] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic embedding for temporal knowledge graph completion. In *AAAI*, 2020.

[14] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[15] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[16] Georgios Gousios and Andy Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 368–371, 2014.

[17] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[18] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thir-*

teenth International Conference on Artificial Intelligence and Statistics, pages 297–304, 2010.

[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[20] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[21] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[22] Christoph Hannebauer, Michael Patalas, Sebastian Stünkel, and Volker Gruhn. Automatically recommending code reviewers based on their expertise: An empirical comparison. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 99–110, 2016.

[23] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[24] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019.

[25] Jing Jiang, Jia-Huan He, and Xue-Yuan Chen. Coredevrec: Automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology*, 30(5):998–1016, 2015.

[26] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. Why and how developers fork what from whom in github. *Empirical Software Engineering*, 22(1):547–578, 2017.

[27] Jing Jiang, David Lo, Jiateng Zheng, Xin Xia, Yun Yang, and Li Zhang. Who should make decision on this pull request? analyzing time-decaying relationships and file

similarities for integrator prediction. *Journal of Systems and Software*, 154:196–210, 2019.

[28] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in neural information processing systems*, pages 4284–4295, 2018.

[29] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

[30] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[31] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 291–302. IEEE, 2016.

[32] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[33] Boris Knyazev, Carolyn Augusta, and Graham W Taylor. Learning temporal attention in dynamic graphs with bilinear interactions. *arXiv preprint arXiv:1909.10367*, 2019.

[34] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. *arXiv preprint arXiv:1806.07297*, 2018.

[35] Julien Leblay and Melisachew Wudage Chekol. Deriving validity time in knowledge graph. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 1771–1776, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356404. doi: 10.1145/3184558.3191639. URL https://doi.org/10.1145/3184558.3191639.

[36] Kalev Leetaru and Philip A. Schrodt. Gdelt: Global data on events, location, and tone. *ISA Annual Convention*, 2013.

[37] Zhifang Liao, Haozhi Jin, Yifan Li, Benhong Zhao, Jinsong Wu, and Shengzong Liu. Devrank: Mining influential developers in github. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[38] Antonio Lima, Luca Rossi, and Mirco Musolesi. Coding together at scale: Github as a collaborative social network. In *Eighth international AAAI conference on weblogs and social media*, 2014.

[39] Hailun Lin, Yong Liu, Weiping Wang, Yinliang Yue, and Zheng Lin. Learning entity and relation embeddings for knowledge resolution. *Procedia Computer Science*, 108: 345–354, 2017.

[40] Chao Liu, Dan Yang, Xiaohong Zhang, Baishakhi Ray, and Md Masudur Rahman. Recommending github projects for developer onboarding. *IEEE Access*, 6:52082–52094, 2018.

[41] Justin Middleton, Emerson Murphy-Hill, Demetrius Green, Adam Meade, Roger Mayer, David White, and Steve McDonald. Which contributions predict whether developers are accepted into github teams. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 403–413. IEEE, 2018.

[42] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

[43] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[44] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. Going farther together: The impact of social capital on sustained participation in open source. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 688–699. IEEE, 2019.

[45] Mitch Rees-Jones, Matthew Martin, and Tim Menzies. Better predictors for issue lifetime. *arXiv preprint arXiv:1702.07735*, 2017.

[46] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[47] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[48] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[49] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[50] Dale J Shpak. A weighted-least-squares matrix decomposition method with application to the design of two-dimensional digital filters. In *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, pages 1070–1073. IEEE, 1990.

[51] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. Acceptance factors of pull requests in open-source projects. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1541–1546, 2015.

[52] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[53] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering*, 24(3):1369–1417, 2019.

[54] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[55] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, 2015.

[56] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3462–3471. JMLR. org, 2017.

[57] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*, 2019.

[58] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*, 2016.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[60] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Aaai*, volume 14, pages 1112–1119. Citeseer, 2014.

[61] Wenyuan Xu, Xiaobing Sun, Jiajun Hu, and Bin Li. Repersp: recommending personalized software projects on github. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 648–652. IEEE, 2017.

[62] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[63] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74:204–218, 2016.

A

Fusion Performance Comparison

Figure A.1 and A.2 provide more insights into the performance comparison of fusion models over consecutive months on all repositories.

Fusion Performance Comparison

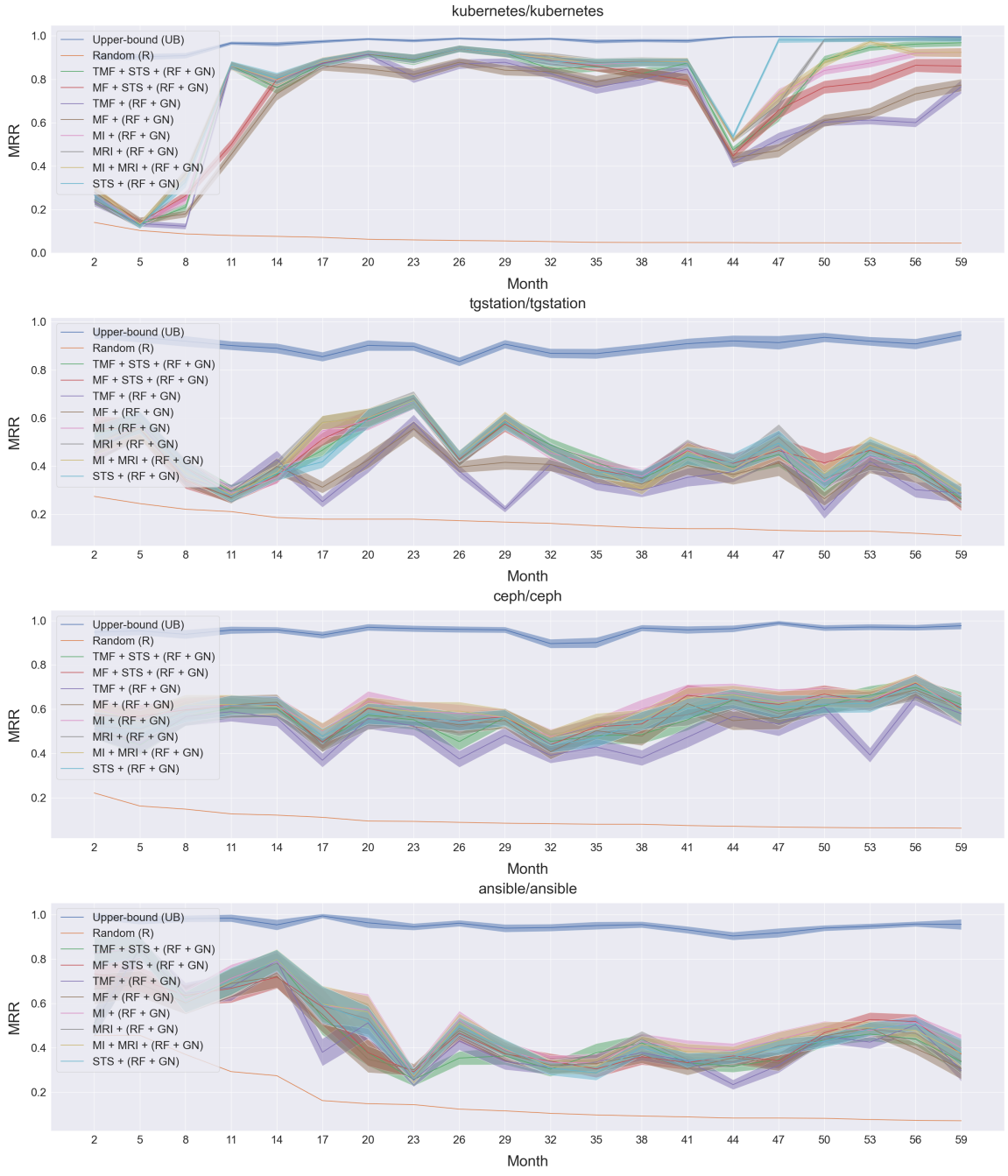

Figure A.1: Gaussian naive bayes fusion baselines performance over all datasets. The area around each line indicates its error bars.

Fusion Performance Comparison

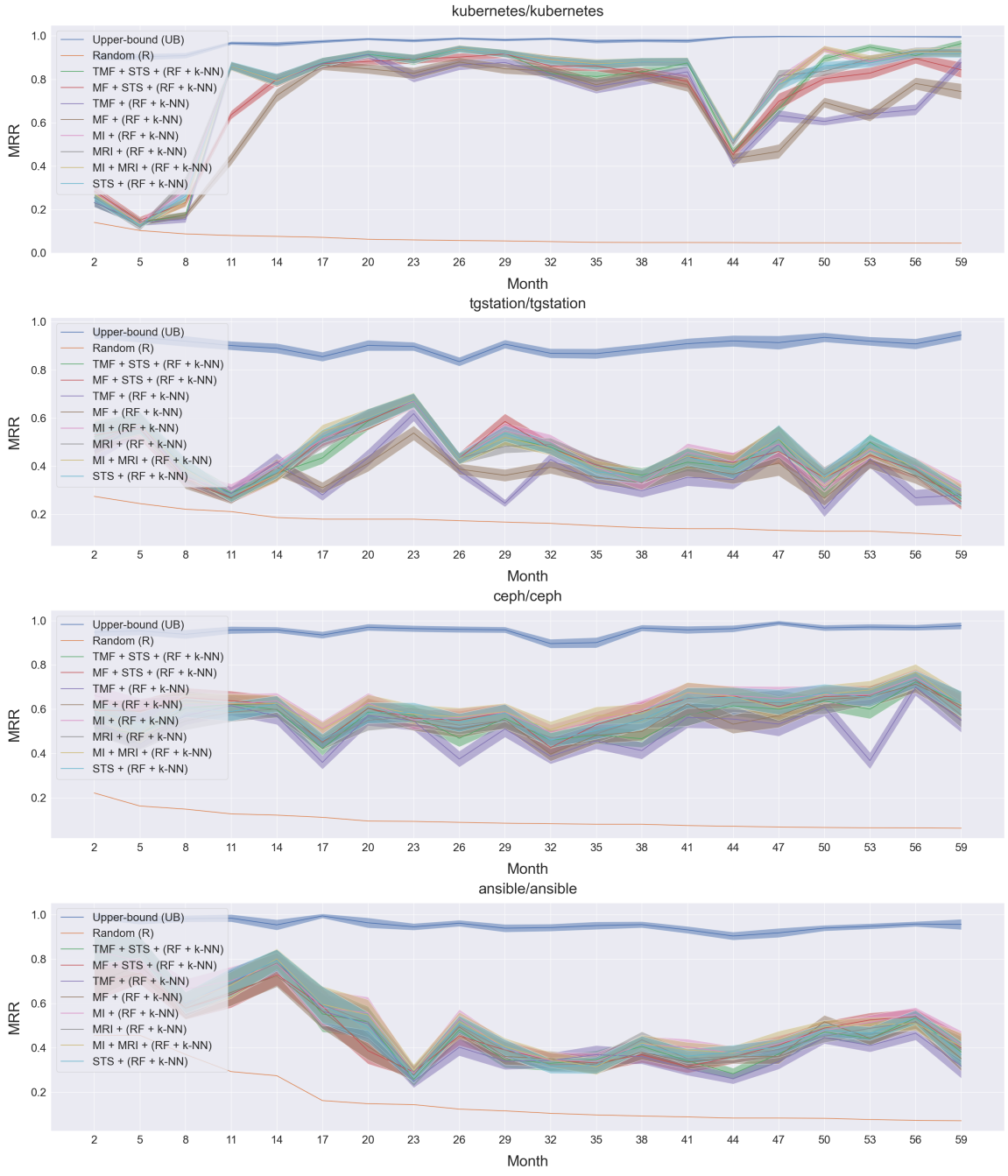

Figure A.2: k-NN fusion baselines performance over all datasets. The area around each line indicates its error bars.