Monte Carlo Algorithms for Nonlinear Filtering, Bayesian Graph Neural Networks, and Probabilistic Forecasting

Soumyasundar Pal

Department of Electrical & Computer Engineering
McGill University
Montréal, Québec, Canada
May 22, 2022

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

Abstract

Computational Bayesian inference has numerous applications in many branches of signal processing and machine learning. Bayesian techniques allow for principled modeling of uncertainty in the design of inference algorithms and offer better empirical performance than their non-Bayesian counterparts in diverse problem settings. However, the posterior distribution of the quantity of interest is not analytically tractable in most practical inference tasks. Approximating the posterior distribution using Monte Carlo methods provides an avenue for implementing Bayesian approaches in these cases. Such approximations often require tremendous computational efforts and scale poorly to complex, high-dimensional settings. Designing scalable and computationally efficient Monte Carlo methods for Bayesian inference is thus of paramount importance in many tasks, and is the main topic of this thesis. The main novel contributions of this thesis can be organized into the following three categories.

First, we address the task of sequential inference of the state of a hidden Markov model in the presence of Gaussian mixture distributed noises. In this setting, the filtering distribution is multi-modal and existing techniques fall short in providing accurate state estimates. We propose several particle flow based algorithms which are suitable for this scenario and offer significant improvement compared to the current *state-of-the-art* filtering techniques.

Second, we develop a general Bayesian Graph Convolutional Neural Network (BGCN) framework and apply it in a semi-supervised node classification problem. Viewing the observed graph as a realization from a parametric family of random graphs, the Bayesian approach targets inference of the joint posterior of the random graph parameters and the node labels. We also propose an extension of the BGCN by incorporating a non-parametric graph inference approach, which extends the applicability of the Bayesian framework to other learning tasks beyond node classification.

Third, we tackle a probabilistic spatio-temporal forecasting task, where utilizing the spatial relationships among the time-series is beneficial for accurate forecasting. We treat the time-series data as a random realization from a nonlinear state-space model and target Bayesian inference of the hidden states for probabilistic forecasting. Particle flow is used as the tool for posterior inference of the states, due to its effectiveness in complex, high-dimensional scenarios. Our approach provides a better characterization of uncertainty while maintaining comparable accuracy to the *state-of-the-art* point forecasting methods.

Abrégé

L'inférence bayésienne computationnelle a de nombreuses applications dans plusieurs branches du domaine de traitement du signal et de l'apprentissage automatique. Les techniques bayésiennes permettent une modélisation fondée sur des principes de l'incertitude dans la conception des algorithmes d'inférence et offrent de meilleures performances empiriques que leurs homologues non bayésiennes dans divers contextes. Cependant, la distribution postérieure de la quantité d'intérêt n'est pas traitable analytiquement dans la plupart des tâches d'inférence pratiques. L'approximation de la distribution a postériori à l'aide des méthodes de Monte Carlo permet de mettre en œuvre des approches bayésiennes pour ces cas-ci. De telles approximations nécessitent souvent d'énormes efforts de calcul et s'adaptent mal à des paramètres complexes de grande dimension. La conception de méthodes Monte Carlo évolutives et efficaces en termes de calcul pour l'inférence bayésienne est donc d'une importance primordiale dans de nombreuses tâches, et constitue le sujet principal de cette thèse. Les principales contributions de cette thèse peuvent être classées dans les trois catégories suivantes.

Premièrement, nous abordons la tâche d'inférence séquentielle de l'état d'un modèle de Markov caché en présence de bruits distribués selon un mélange gaussien. Dans ce cadre, la distribution de filtrage est multimodale et les techniques existantes ne parviennent pas à fournir des estimations d'état précises. Nous proposons plusieurs algorithmes basés sur le flux de particules qui conviennent à ce scénario et offrent une amélioration significative par rapport à *l'état de l'art* actuel des techniques de filtrage.

Deuxièmement, nous développons le cadre général Bayesian Graph Convolutional Neural Network (BGCN) et l'appliquons à un problème de classification de nœuds semi-supervisé. En considérant le graphe observé comme une réalisation d'une famille paramétrique de graphes aléatoires, l'approche bayésienne cible l'inférence de la distribution conjointe a postériori des paramètres du graphe aléatoire et des étiquettes de nœuds. Nous proposons également une extension du BGCN en incorporant une approche d'inférence de graphe non paramétrique, qui étend l'applicabilité du cadre bayésien à d'autres tâches d'apprentissage au-delà de la classification des nœuds.

Troisièmement, nous abordons une tâche de prévision spatio-temporelle probabiliste, où l'utilisation des relations spatiales entre les séries chronologiques permet une prévision précise. Nous traitons les données de séries chronologiques comme une réalisation aléatoire

à partir d'un modèle espace-état non linéaire et l'inférence bayésienne cible des états cachés pour la prévision probabiliste. Le flux de particules est utilisé comme outil pour l'inférence des états a postériori, en raison de son efficacité dans des scénarios complexes de grande dimension. Notre approche permet une meilleure caractérisation de l'incertitude tout en maintenant une précision comparable à *l'état de l'art* des méthodes de prévision ponctuelle.

Acknowledgements

First and foremost, I would like to express my sincere gratitude toward my supervisor Prof. Mark Coates. I am tremendously grateful for his continuous support, for many in-depth discussions, and for his tireless efforts in supervising my research. As a student in his research group, I got many opportunities to learn valuable research skills. His insightful feedback and invaluable guidance contributed greatly in my progress as a graduate student. I am also truly grateful for the thoughtful feedback, that I have received from my graduate committee members, Prof. David Stephens and Prof. Ioannis Psaromiligkos, during several presentations in course of my PhD.

I have been extremely fortunate to work with many talented researchers from McGill and Huawei, including Dr. Yunpeng Li, Dr. Deniz Üstebay, Florence, Liheng, Yingxue, Antonios, Arezou, Fatemeh, Saber, Yishi and Jianing. Collaborative research has always been a learning experience for me, and I am grateful to all of you for being fantastic collaborators.

The lab has always been a stimulating environment, thanks to everyone in the group. The friendship and support of the labmates have been a key factor in dealing with the pandemic-induced stress. I also appreciate the support and guidance provided by the department of Electrical and Computer Engineering of McGill University.

Last but not least, I am forever indebted to my family for their relentless love, support, and encouragement throughout my life.

Contents

List of Tables

List of Figures

List of Acronyms

TPU	Tensor processing unit
HMM	Hidden Markov model
SSM	State-space model
SMC	Sequential Monte Carlo
MCMC	Markov chain Monte Carlo
SIS	Sequential importance sampling
SISR	Sequential importance sampling with resampling
ESS	Effective sample size
BPF	Bootstrap particle filter
APF	Auxiliary particle filter
UPF	Unscented particle filter
DH	Daum-Huang
EDH	Exact Daum-Huang
LEDH	Localized exact Daum-Huang
PFPF	Particle flow particle filter
GSMC	Guided Sequential Monte Carlo
GPFIS	Gaussian particle flow importance sampling
SMCMC	Sequential Markov chain Monte Carlo
SIMCMC	Sequentially interacting Markov chain Monte Carlo
HMC	Hamiltonian Monte Carlo
mHMC	manifold Hamiltonian Monte Carlo
MAP	Maximum a posteriori
MLP	Multi-layer perceptron
CNN	Convolutional neural networks
RNN	Recurrent neural networks
GNN	Graph neural network
GCN	Graph convolutional networks
VGAE	Variational graph autoencoder
DGLFRM	Deep generative latent feature relational model
SBM	Stochastic block model
MMSBM	Mixed membership stochastic block model

AR	Auto-regressive
ARMA	Auto-regressive moving average
ARIMA	Auto-regressive integrated moving average
SARIMA	Seasonal auto-regressive integrated moving average
LSTM	Long short-term memory
GRU	Gated recurrent unit
TRMF	Temporal regularized matrix factorization
N-BEATS	Neural basis expansion analysis for time-series
MQRNN	Multi-quantile recurrent neural networks
TGCN	Temporal graph convolutional network
DCRNN	Diffusion convolutional recurrent neural network
AGCRN	Adaptive graph convolutional recurrent network
STGCN	Spatio-temporal graph convolutional network
LSGCN	Long short-term graph convolutional network
MTGNN	Multivariate Time-series forecasting with Graph Neural Network
SLCNN	Structure learning convolutional neural network
STSGCN	Spatial-temporal synchronous graph convolutional network
ASTGCN	Attention spatial-temporal graph convolutional networks
GMAN	Graph multi-attention network
STGRAT	Spatio-Temporal GRaph ATtention
FC-GAGA	Fully Connected GAted Graph Architecture

Chapter 1
Introduction

As per the Bayesian interpretation, a degree of belief in an event is expressed via probability. This is considerably different from the 'frequentist' approach, which views probability as the limit of the relative frequency of an event, after a large number of trials have been conducted. In the Bayesian setting, the unknown parameters of a statistical model are considered random variables and a 'prior' distribution is adopted to the model parameters to assign one's beliefs about them before observing any data. Any background knowledge, such as information from previous experiments and/or subjective assessment of an expert can be utilized to determine a suitable prior over the parameters. In light of the observed data, one updates this belief from the 'prior' to the 'posterior' distribution of the parameters by applying Bayes' theorem. Prediction from the model for new data is obtained by computing the posterior predictive distribution, which involves marginalization with respect to the posterior distribution over the model parameters.

Aside from interesting theoretical connotations [1] that go back to the work of de Finetti, the Bayesian framework offers several advantages. Incorporation of the uncertainty associated with model parameters into the statistical inference task and gauging the uncertainty in the predictions of a statistical model are principled yet seamlessly natural in the Bayesian context. Computation of the posterior distribution of parameters is inherently sequential, i.e., in settings where data is observed in a streaming fashion, the current posterior distribution serves as the prior to account for the effect of the next observation. In addition, handling of nuisance parameters is elegantly accomplished by marginalization. The Bayesian approach also provides a straightforward probabilistic interpretation of the confidence intervals without resorting to asymptotics. In view of these, an eminent statistician and a vocal proponent of Bayesian approaches, Dennis V. Lindley, claimed, "the only good statistics is Bayesian statistics" [2], and predicted "a Bayesian 21st century" in 1975.

There are two key ingredients of the Bayesian approach: a) computing the posterior distribution; and b) obtaining the posterior predictive distribution. Except for simple settings, such as the parametric models with conjugate priors, the posterior distribution of interest does not have a closed form in most practical problems. Computing the posterior

predictive density requires evaluation of a potentially high-dimensional integral, depending on the complexity of the model. Practical implementation of Bayesian methodology is thus not straightforward and often requires approximation. Monte Carlo methods offer an elegant solution to both of these problems. The posterior distribution is approximated by a set of Monte Carlo samples, which are subsequently used for estimating the predictive distribution.

There are several classes of Monte Carlo methods, such as rejection sampling [3], importance sampling [4], and Markov Chain Monte Carlo [5]. The choice of a suitable algorithm typically depends on the complexity of an inference task and the availability of the computational resources. The rapid improvement of both hardware and software suitable for large scale computations in recent decades has been a key driving force of the research for the development of novel Monte Carlo algorithms and their applications to solve challenging, practical problems. As an illustrative example, one can observe the historical evolution of Bayesian neural networks. Earlier approaches [6–8] were scalable to only a few hidden layers because of the limited computing power available thirty years ago, whereas training of Bayesian deep learning architectures with millions of parameters has been carried out recently using several clusters containing hundreds of *tensor processing units* (TPUs) [9].

Despite their practical utility, generic Monte Carlo methods often perform poorly when used to solve challenging inference problems in signal processing and machine learning applications. They often have extremely high computational requirements and fail to scale to high-dimensions. The design of scalable and computationally efficient techniques for Bayesian inference thus requires careful consideration and is an active area of research. In this thesis, we focus on the development of novel Monte Carlo algorithms for three different tasks: high-dimensional nonlinear filtering, learning from graph structured data, and probabilistic spatio-temporal forecasting.

Effective and efficient online learning of high-dimensional states is an important task in many domains where we need to regularly update our knowledge by processing a deluge of streaming data. Relevant applications include robotic navigation [10], multi-target tracking [11], and weather forecasting [12]. In this setting, we aim to estimate the states of a *hidden Markov model* (HMM) from a sequence of observations. Since the state-transition

and the observation process of the HMM are noisy, exact estimation is infeasible and a Bayesian framework is adopted for recursive computation of the filtering distribution, i.e., the marginal posterior distribution of the states. However, except for idealized scenarios, such as a linear-Gaussian model, filtering cannot be performed analytically. Use of particle filters [13] has become a standard approach for state estimation in nonlinear systems. However, their performance usually deteriorates if the dimension of the state space is high or the measurements are highly informative [14, 15]. Although there have been multiple proposals to address this issue [14–19], the methods rely on the posterior having a special structure or are computationally expensive. Particle flow filters [20, 21] can achieve impressive performance for a much reduced computational overhead. Instead of sampling, particles are migrated from the prior to the posterior by identifying and solving differential equations that link these two distributions. The approximations needed to implement these filters can lead to particles not being a genuine sample from the posterior. To address this, several recent algorithms combine particle flow and particle filtering [22–26]. Another class of algorithms are the sequential Markov Chain Monte Carlo (SMCMC) [15, 27] methods, which use a Metropolis-Hastings (MH) accept-reject approach to improve filtering performance in high dimensions.

A major limitation of many of these existing approaches is that their design relies on implicit uni-modality assumptions (e.g., Gaussian density or log-concavity) regarding the predictive and the posterior distributions [15, 21, 23]. As a result, they exhibit poor performance in tracking multi-modal posteriors. Several approaches [28–31], which can deal with multi-modality, either scale poorly to high dimensions or break down in the presence of high nonlinearity. In Chapter 3 of this thesis, we address a setting where the dynamic and measurement models can be approximated by a nonlinearity with additive noise distributed according to a Gaussian mixture. We develop several particle flow based filtering algorithms, which are demonstrably superior to existing techniques in high-dimensional state estimation.

A significant body of research focuses on using neural networks to analyze structured data when there is an underlying graph describing the relationship between data items. Alleviating the difficulties associated with the computational inefficiency of the earlier approaches [32–34], numerous *graph neural network* (GNN) [35, 36] models have been proposed in recent years for addressing various graph based learning tasks such as node

classification [37, 38] and link prediction [39, 40]. These approaches process the observed graph as if it depicts the true relationship among the nodes. In practice, the observed graphs are often formed based on imperfect observations and approximate modelling assumptions. Spurious edges might be present and important links might be deleted. The vast majority of existing algorithms cannot take the uncertainty of the graph structure into account during training as there is no mechanism for removing spurious edges and/or adding informative edges in the observed graph.

Several algorithms that do address this uncertainty by incorporating a graph learning component have been proposed recently [41–45]. However, these methods either use variational approximation that introduces a bias in the approximate posterior of the graph or have limited applicability, since these models focus only on the task of node classification. In Chapter 4 of this thesis, we consider a generally applicable Bayesian framework for node- and edge-level learning tasks by viewing the 'true' graph as a random quantity. We target posterior inference of the graph topology, and combine this graph inference procedure with the training of graph neural network architectures to account for the uncertainty arising from the graph structure.

We also consider a spatio-temporal forecasting task, which has numerous applications in analyzing wireless, traffic, and financial networks. For example, accurate forecasting of car speed at different roads of a city can potentially improve traffic management and reduce congestion. This ensures faster commute for the drivers and helps in reducing air pollution. In this case, there are useful similarities among the time-series at adjoining roads. Suppose congestion is observed at a particular junction of a road. Then, one can expect a similar phenomenon to happen at the preceding segments that lead to that junction. If an algorithm can model these type of spatial dependencies effectively, then it can achieve improved forecasting performance. Such spatial relationships are often encoded in the form of a graph.

Many classical statistical models [46, 47] disregard this graph completely and fall short in handling the complexity and high non-linearity present in time-series data. Recent advances in deep learning [48–51] allow for better modelling of spatial and temporal dependencies. While most of these models focus on obtaining accurate point forecasts, they do not characterize the prediction uncertainty, which is crucial for intelligent use of the

obtained forecasts in other downstream tasks. On the other hand, existing probabilistic forecasting models [52–54] are not equipped to exploit the spatial relationships among the time-series. In order to address these issues, we cast the forecasting task in a Bayesian framework using a graph cognizant, nonlinear state-space model in Chapter 5 of this thesis and derive a probabilistic forecasting technique that can incorporate the graph structure. We conduct experiments on several time-series datasets to demonstrate the effectiveness of the proposed algorithm.

1.1 Thesis Organization and Contributions

We describe the organization of the thesis and summarize the main technical contributions in this section. The material presented in this thesis has been published in various journals and conferences such as IEEE Transactions on Signal Processing, 2019 [55], the Advancement of Artificial Intelligence Conference (AAAI), 2019 [56], the Conference on Uncertainty in Artificial Intelligence (UAI), 2020 [57], and International Conference on Machine Learning (ICML), 2021 [58].

- *Chapter 2 - Background Material and Literature Review:* We present a comprehensive coverage of background material required for this thesis. The first part reviews the common sequential inference techniques in a state-space model and the recent developments in particle filter, particle flow, particle flow particle filter, and sequential MCMC (SMCMC) methods. The second part of the chapter is devoted to graph neural networks and some material concerning generative models for graphs. We provide a literature review of time-series forecasting models in the third part.
- *Chapter 3 - Sequential Inference in the Presence of Gaussian Mixture Noise Models:* In this chapter, we propose three novel algorithms for sequential inference in a state-space model when both process and measurement noises are distributed as mixtures of Gaussians. These algorithms are compared to existing techniques in challenging high-dimensional filtering problems with multi-modal posteriors. We introduce Particle Flow for Gaussian Mixture Models (PF-GMM) [59], which generalizes the particle flow approach to multi-modal posterior distributions. We then propose a novel particle filtering algorithm, called the Particle Flow Particle Filter for Gaussian Mixture Models (PFPF-GMM) [60], for the same problem setting to mitigate the effect of the improper

model assumptions and numerical approximations required for computing the flow. Finally, we propose the Sequential Markov Chain Monte Carlo method for Gaussian Mixture Models (SMCMC-GMM) [55], which uses a modified version of invertible particle flow to construct a suitable multi-modal proposal distribution in the joint draw step of the SMCMC algorithm.

- *Chapter 4 - Bayesian Graph Convolutional Neural Networks:* This chapter presents a Bayesian framework for learning on graphs. We view the observed graph as a realization from a parametric random graph model and target inference of the joint posterior of the random graph parameters and the GNN weights [56]. The resulting algorithm is compared to several baseline techniques in a transductive node classification task and in the face of adversarial attacks. We then propose a novel non-parametric graph model for constructing the posterior distribution of graph adjacency matrices in the Bayesian framework [57]. This allows us to extend the proposed methodology to other graph related learning tasks beyond node classification.
- *Chapter 5 - RNN with Particle Flow:* In this chapter, we introduce a novel Bayesian framework for probabilistic spatio-temporal forecasting [58]. We view the spatio-temporal data as a sequence of random observations from a graph cognizant, nonlinear state-space model and cast the forecasting task in the Bayesian framework. The effectiveness of particle flow [21] in complex, high-dimensional settings motivates its use as the Monte Carlo method for posterior inference of the hidden states. Thorough experimentation on several real world time-series datasets demonstrates the benefits of the proposed methodology.
- *Chapter 6 - Conclusions and Future Work:* In this chapter, we summarize the main contributions of this thesis and discuss the main results. In addition, we identify potential future research directions, including several strategies for improving the methodologies developed in this thesis.

1.1.1 Publications and Contributions of Collaborators

- *Chapter 3*
  - S. Pal and M. Coates, "Gaussian sum particle flow filter," in *Proc. IEEE Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.*, Curacao, The

Netherlands, Dec. 2017, pp. 1–5. [59]

Prof. Mark Coates proposed the idea of extending particle flow approaches to track multi-modal posterior distributions. I derived the proposed PF-GMM algorithm, implemented it along with other baseline algorithms, and evaluated their performance in several numerical simulation setups.

– S. Pal and M. Coates, "Particle flow particle filter for Gaussian mixture noise models," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, Calgary, Canada, Apr. 2018, pp. 4249–4253. [60]

Prof. Mark Coates proposed the idea of viewing the hidden Markov model with Gaussian mixture noises as a switching state-space model for facilitating the development of a particle flow particle filter. I derived and implemented the proposed PFPF-GMM algorithm and compared its performance with several baseline methods.

– Y. Li, S. Pal, and M. Coates, "Invertible particle flow-based sequential MCMC with extension to Gaussian mixture noise models," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2499–2512, May 2019. [55]

Dr. Yunpeng Li and Prof. Mark Coates published a conference paper [61] on incorporating invertible particle flow into the joint draw step of Sequential MCMC. In this journal paper, we provided a more detailed description of the proposed approach, presented more computationally efficient algorithms, and proposed a sequential MCMC method with mixture model-based flow for efficient exploration of multi-modal distributions. We also derived theoretical results regarding asymptotic convergence of the algorithms. I proposed the novel SmHMC-GMM (LEDH) algorithm, conducted its implementation, and participated in the derivation of the theoretical results. I was co-first author (with Dr. Li) in recognition of our joint effort in developing the methodology.

- *Chapter 4*
  - Y. Zhang, S. Pal, M. Coates, and D. Üstebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proc. AAAI Conf. Artificial Intell.*, Honolulu, HI, USA, Feb. 2019, pp. 5829–5836. [56]

Prof. Mark Coates proposed the idea of incorporating the uncertainty associated with the graph topology into *graph convolutional networks* (GCN) [37, 62] and provided a derivation of the resulting algorithm. I was a co-first author (with Yingxue Zhang) and my responsibilities were implementation of the stochastic gradient maximum a posteriori (MAP) estimation algorithm of the assortative-Mixed Membership Stochastic Block Model (a-MMSBM) parameters and integrating it with the Graph Convolutional Network (GCN) module. Yingxue Zhang helped in designing the experimental setup, ran several baseline algorithms, and conducted the experiment to test the adversarial robustness of the proposed algorithm. Dr. Deniz Üstebay and Prof. Mark Coates provided valuable advice on designing the experiments and contributed greatly in writing the manuscript.

- S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, "Non-parametric graph learning for Bayesian graph neural networks," in *Proc. Conf. Uncertainty in Artificial Intell.*, Virtual, Aug. 2020. [57]

  Prof. Mark Coates asked me to consider extension of the Bayesian learning framework to other learning tasks beyond node classification. I derived and implemented the non-parametric graph inference based node classification and link prediction algorithms. Florence Regol contributed in designing the experimental setups and data visualization. Using my implementation of the non-parametric graph inference algorithm, Saber Malekmohammadi and Yishi Yu conducted the recommendation system experiments (not included in this thesis for conciseness). Yingxue Zhang and Prof. Mark Coates supervised the research plan and edited the manuscript.

- *Chapter 5*

  - S. Pal, L. Ma, Y. Zhang, and M. Coates, "RNN with particle flow for probabilistic spatio-temporal forecasting," in *Proc. Int. Conf. Machine Learning*, Virtual, Jul. 2021. [58]

    Prof. Mark Coates proposed the initial idea of conducting posterior inference of the RNN states using particle flow. I derived the proposed algorithms and implemented them along with most of the baselines. Liheng Ma conducted

implementation of the rest of the baseline algorithms and helped in data visualization. Yingxue Zhang and Prof. Mark Coates provided valuable advice on designing the experiments and contributed immensely in writing the manuscript.

Chapter 2
Background Material and Literature Review

In this chapter, we cover the key background material and literature review for the three main topics addressed in this thesis. In Section 2.1, we introduce notation, state the filtering task, and review several classes of sequential inference techniques, such as particle filter, particle flow and sequential MCMC (SMCMC) algorithms. An introduction to graph neural networks and some relevant material on generative models for graphs is provided in Section 2.2. Section 2.3 presents a detailed review of the existing work on time-series forecasting algorithms.

2.1 Sequential Inference Techniques

In this section, we introduce the hidden Markov model and state the sequential inference task in the Bayesian framework. Subsequently, we review several classes of filtering techniques such as Kalman filters, particle filters (a.k.a. sequential Monte Carlo), and sequential *Markov Chain Monte Carlo* (MCMC) methods. Particle flow filters, which exhibit impressive performance for many high-dimensional filtering problems in nonlinear and non-Gaussian models, are discussed briefly. We also provide a review of advanced particle filtering and sequential MCMC approaches, which are based on particle flow.

2.1.1 Hidden Markov Models

In many settings such as robotic learning [10], financial modelling [63], multi-target tracking [11], and weather forecasting [12], online inference of evolving states by processing a sequence of observed data is essential. The *hidden Markov model* (HMM) is a natural choice as a statistical model to explain the state transition and the observation process in these cases. In this model, the dynamics of the unobserved (hidden) state are governed by the Markov property; that is, the current state is conditionally independent of the past state trajectory given the most recent state. There is another process, called the observation or measurement process, which is modelled as independent of all past states

and measurements conditioned on the current state. Let x_0 denote the initial state before any measurement arrives and x_k and z_k represent the hidden state and the measurement at time k, respectively. The hidden Markov model is described as:

$$x_0 \sim p(x_0), \tag{2.1}$$

$$x_k = g_k(x_{k-1}, v_k) \quad \text{for } k \geqslant 1\,, \tag{2.2}$$

$$z_k = h_k(x_k, w_k) \quad \text{for } k \geqslant 1\,. \tag{2.3}$$

Here $p(x_0)$ is an initial probability density function, $g_k : \mathbb{R}^{d_x} \times \mathbb{R}^{d_v} \to \mathbb{R}^{d_x}$ is the state-transition function of the unobserved state $x_k \in \mathbb{R}^{d_x}$, $z_k \in \mathbb{R}^{d_z}$ is the measurement generated from the state x_k through a potentially nonlinear measurement model $h_k : \mathbb{R}^{d_x} \times \mathbb{R}^{d_w} \to \mathbb{R}^{d_z}$. $v_k \in \mathbb{R}^{d_v}$ is the process noise, and $w_k \in \mathbb{R}^{d_w}$ is the measurement noise. v_k and w_k are assumed to be mutually independent and independent of the initial state x_0.

Based on this model, the discrete time filtering task is to compute the marginal posterior distribution of the state trajectory $p(x_k|z_1, \cdots, z_k)$ recursively with time k. For conciseness, we use $x_{a:b}$ to denote the set $\{x_a, x_{a+1}, \cdots, x_b\}$ and $z_{a:b}$ to denote the set $\{z_a, z_{a+1}, \cdots, z_b\}$, where a and b are integers and $a < b$. The posterior distribution $p(x_k|z_{1:k})$ summarizes the uncertainty in the state x_k given all measurements up to k-th time step and is used for state estimation.

2.1.2 Kalman Filter

If the HMM is linear-Gaussian, i.e., the initial state x_0 is normally distributed and both the dynamic and measurement models are linear with additive Gaussian noises, the predictive distribution and the posterior distribution at each time step are Gaussian as well. In that case, the Kalman filter [64, 65] provides a closed-form solution to obtain $p(x_k|z_{1:k})$ from $p(x_{k-1}|z_{1:k-1})$ recursively. Suppose the linear-Gaussian HMM is represented as:

$$x_0 \sim \mathcal{N}(x_0; \mu_0, P_0)\,, \tag{2.4}$$

$$x_k = G_k x_{k-1} + v_k \quad \text{for } k \geqslant 1\,, \tag{2.5}$$

$$z_k = H_k x_k + w_k \quad \text{for } k \geqslant 1\,. \tag{2.6}$$

Here, $\mu_0 \in \mathbb{R}^{d_x}$ and $P_0 \in \mathbb{R}^{d_x \times d_x}$ are the mean and covariance matrix of x_0. $G_k \in \mathbb{R}^{d_x \times d_x}$ and $H_k \in \mathbb{R}^{d_z \times d_x}$ denote the state transition and measurement matrices respectively. $v_k \sim \mathcal{N}(\mathbf{0}, Q_k)$ and $w_k \sim \mathcal{N}(\mathbf{0}, R_k)$ are zero-mean additive Gaussian noises.

Suppose the normal posterior distribution at time $k-1$, $p(x_{k-1}|z_{1:k-1})$, has mean $\mu_{k-1|k-1}$ and covariance matrix $P_{k-1|k-1}$. We initialize $\mu_{0|0} = \mu_0$ and $P_{0|0} = P_0$ at $k = 0$. At each subsequent time step $k \geqslant 1$, the Kalman filter performs the following two steps:

Predict step computes the predictive posterior distribution $p(x_k|z_{1:k-1})$ at time step k from the posterior distribution $p(x_{k-1}|z_{1:k-1})$ of the previous time step as follows:

$$p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1})\, dx_{k-1}\,. \tag{2.7}$$

Since both terms of the integrand in eq. (2.7) are Gaussian in a linear-Gaussian HMM, the predictive posterior distribution is also normal and its mean $\mu_{k|k-1}$ and covariance matrix $P_{k|k-1}$ can be computed in closed form as follows:

$$\mu_{k|k-1} = G_k\mu_{k-1|k-1}\,, \tag{2.8}$$
$$P_{k|k-1} = G_kP_{k-1|k-1}G_k^T + Q_k\,. \tag{2.9}$$

Update step uses the predictive posterior distribution $p(x_k|z_{1:k-1})$ as the prior in time step k in conjunction with the observation likelihood $p(z_k|x_k)$ and applies Bayes' theorem to compute the current step posterior $p(x_k|z_{1:k})$ as follows:

$$p(x_k|z_{1:k}) = \frac{p(x_k|z_{1:k-1})p(z_k|x_k)}{\int p(\widetilde{x}_k|z_{1:k-1})p(z_k|\widetilde{x}_k)\, d\widetilde{x}_k}\,. \tag{2.10}$$

For a linear-Gaussian HMM, $p(x_k|z_{1:k}) = \mathcal{N}(x_k; \mu_{k|k}, P_{k|k})$ and the recursive computation for $\mu_{k|k}$ and $P_{k|k}$ are given as:

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1}\,, \tag{2.11}$$
$$\mu_{k|k} = \mu_{k|k-1} + K_k(z_k - H_k\mu_{k|k-1})\,, \tag{2.12}$$
$$P_{k|k} = (I - K_kH_k)P_{k|k-1}\,. \tag{2.13}$$

2.1.3 Extended Kalman Filter

If the HMM has nonlinearity in its state-transition and/or measurement model and the process and measurement noises are additive, then it is represented as:

$$x_k = g_k(x_{k-1}) + v_k \quad \text{for } k \geqslant 1\,, \tag{2.14}$$

$$z_k = h_k(x_k) + w_k \quad \text{for } k \geqslant 1\,. \tag{2.15}$$

In this case, the posterior distribution of the state is non-Gaussian and cannot be computed in closed form in general using the Kalman filter in Section 2.1.3. A first order Taylor expansion based approximation of the dynamic and measure models is employed in the Extended Kalman Filter (EKF) [65] to obtain a Gaussian approximation of the posterior distribution.

Formally, linearization of the dynamic model at the mean of the posterior distribution of the previous state yields:

$$G_k = \left.\frac{\partial g_k(x)}{\partial x}\right|_{x=\mu_{k-1|k-1}}\,. \tag{2.16}$$

Similarly, the measurement model is linearized at the mean of the predictive posterior distribution of the state to obtain

$$H_k = \left.\frac{\partial h_k(x)}{\partial x}\right|_{x=\mu_{k|k-1}}\,. \tag{2.17}$$

Subsequently, G_k and H_k are used in eq. (2.9), (2.11), and (2.13). The computation of the approximate mean of the predictive posterior distribution in eq. (2.8) is modified as:

$$\mu_{k|k-1} = g_k(\mu_{k-1|k-1})\,. \tag{2.18}$$

Similarly, the computation of the approximate posterior mean in eq. (2.12) is changed to

$$\mu_{k|k} = \mu_{k|k-1} + K_k(z_k - h_k(\mu_{k|k-1}))\,. \tag{2.19}$$

Although the EKF is widely used to perform nonlinear filtering, it shows poor performance when the linear approximation of the HMM is severely inaccurate.

2.1.4 Gaussian Sum Filter

In many cases, the predictive posterior distribution and/or the measurement likelihood are multi-modal [29–31] and the Gaussian approximations result in a large estimation error. The Gaussian Sum Filter (GSF) [30] framework considers a setting where both $v_k \sim \sum_{m=1}^{M} \psi_{k,m}\mathcal{N}(\tau_{k,m}, Q_{k,m})$ and $w_k \sim \sum_{n=1}^{N} \beta_{k,n}\mathcal{N}(\zeta_{k,n}, R_{k,n})$, are distributed according to Gaussian mixture models (GMMs).

Suppose the posterior distribution at time step $k-1$ is approximated by L component Gaussian mixture as follows:

$$p(x_{k-1}|z_{1:k-1}) \approx \sum_{\ell=1}^{L} \gamma_{k-1,\ell}\mathcal{N}(x_k|\mu_{k-1,\ell}, P_{k-1,\ell})\,. \tag{2.20}$$

The predictive posterior distribution at time k can be approximated as:

$$p(x_k|z_{1:k-1}) \approx \sum_{\ell=1}^{L}\sum_{m=1}^{M} \alpha_{k,\ell m}\mathcal{N}(x_k|\bar{\mu}_{k,\ell m}, \bar{P}_{k,\ell m})\,, \tag{2.21}$$

where

$$\alpha_{k,\ell m} = \gamma_{k-1,l}\psi_{k,m}\,, \tag{2.22}$$

$$\bar{\mu}_{k,\ell m} = g_k(\mu_{k-1,\ell}) + \tau_{k,m}\,, \tag{2.23}$$

$$G_{k,\ell} = \frac{\partial g_k(x)}{\partial x}|_{x=\mu_{k-1,\ell}}\,, \tag{2.24}$$

$$\bar{P}_{k,\ell m} = G_{k,\ell}P_{k-1,\ell}G_{k,\ell}^T + Q_{k,m}\,. \tag{2.25}$$

Resampling of GMM components, as in [30] (see Algorithm 2.1), can be employed for further approximation of this predictive posterior distribution. This ensures that the number of mixture components does not grow exponentially with time k. Suppose the predictive posterior is approximated as an L'-component GMM after component

resampling as follows:

$$p(x_k|z_{1:k-1}) \approx \sum_{\ell'=1}^{L'} \alpha_{k,\ell'} \mathcal{N}(x_k; \bar{\mu}_{k,\ell'}, \bar{P}_{k,\ell'}) \,. \tag{2.26}$$

The likelihood of the observation z_k is given as:

$$p(z_k|x_k) = \sum_{n=1}^{N} \beta_{k,n} \mathcal{N}(z_k; h_k(x_k) + \zeta_{k,n}, R_{k,n}) \,. \tag{2.27}$$

The posterior distribution of the state at time step k can be approximated as:

$$p(x_k|z_{1:k}) \approx \sum_{\ell'=1}^{L'} \sum_{n=1}^{N} \gamma_{k,\ell' n} \mathcal{N}(x_k; \mu_{k,\ell' n}, P_{k,\ell' n}) \,, \tag{2.28}$$

From the interaction of the ℓ'-th component in the predictive posterior and the n-th component of the likelihood, we have the usual extended Kalman filter update:

$$H_{k,\ell'} = \frac{\partial h_k(x)}{\partial x}|_{x=\bar{\mu}_{k,\ell'}} \,, \tag{2.29}$$

$$K_{k,\ell' n} = \bar{P}_{k,\ell'} H_{k,\ell'}^T (H_{k,\ell'} \bar{P}_{k,\ell'} H_{k,\ell'}^T + R_{k,n})^{-1} \,, \tag{2.30}$$

$$\mu_{k,\ell' n} = \bar{\mu}_{k,\ell'} + K_{k,\ell' n}(z_k - \zeta_{k,n} - h_k(\bar{\mu}_{k,\ell'})) \,, \tag{2.31}$$

$$P_{k,\ell' n} = (I - K_{k,\ell' n} H_{k,\ell'}) \bar{P}_{k,\ell'} \,. \tag{2.32}$$

The mixture proportion $\gamma_{k,\ell' n}$ is given as follows:

$$\gamma_{k,\ell' n} = \frac{\xi_{k,\ell' n}}{\Sigma_{i=1}^{L'} \Sigma_{j=1}^{N} \xi_{k,ij}} \,, \tag{2.33}$$

$$\xi_{k,ij} = \alpha_{k,i} \beta_{k,j} \mathcal{N}(z_k; h_k(\bar{\mu}_{k,i}) + \zeta_{k,j}, H_{k,i} \bar{P}_{k,i} H_{k,i}^T + R_{k,j}). \tag{2.34}$$

Since the predictive posterior and likelihood have L' and N Gaussian components respectively, the posterior is composed of $L'N$ components. As in the predict step, resampling of GMM components is performed to reduce the number of components in the

final representation to L. The posterior is then approximated as:

$$p(x_k|z_{1:k}) \approx \sum_{\ell=1}^{L} \gamma_{k,\ell} \mathcal{N}(x_k; \mu_{k,\ell}, P_{k,\ell}) \,. \tag{2.35}$$

Each component of the predictive and posterior distribution follows the extended Kalman filter (EKF) equations. We can recursively track the posterior by employing parallel EKFs. However, in higher dimensions, if g_k or h_k is highly nonlinear, this approach performs poorly.

Algorithm 2.1 Resampling of Gaussian components in a GMM in [30].
Input: GMM $\sum_{i=1}^{M} \alpha_i \mathcal{N}(\mu_i, P_i)$, number of output components $N < M$, threshold $\alpha_{th} > 0$.
Output: GMM $\sum_{j=1}^{N} \beta_j \mathcal{N}(\tilde{\mu}_j, \tilde{P}_j)$.

Sort $\{\alpha_i\}_{i=1}^{M}$ in descending order to obtain $i_1, i_2, \cdots, i_M \in \{1, 2, \cdots, M\}$ such that $\alpha_{i_1} \geqslant \alpha_{i_2} \geqslant \cdots \geqslant \alpha_{i_M}$.
Set $\beta'_j = \alpha_{i_j}$ for $j = 1, \cdots, N$.
if $\beta'_N < \alpha_{th}$ **then**
  for $j = 1, \cdots, N$ **do**
    Sample $k \in \{1, \cdots, N\}$ with probability proportional to $\{\beta'_1, \cdots, \beta'_N\}$.
    Set $\beta_j = \frac{1}{N}$, $\tilde{\mu}_j = \mu_{i_k}$, and $\tilde{P}_j = P_{i_k}$.
  end for
else
  Set $\beta_j = \frac{\beta'_{i_j}}{\sum_{k=1}^{N} \beta'_{i_k}}$, $\tilde{\mu}_j = \mu_{i_j}$, and $\tilde{P}_j = P_{i_j}$, for $j =, \cdots, N$.
end if

2.1.5 Particle Filter

Although there were several similar earlier attempts [66, 67] of using Monte Carlo methods in nonlinear filtering, the seminal work by Gordon et al. in [13] is generally considered to be the first instance of modern particle filters. These constitute a class of algorithms based on *sequential importance sampling* (SIS) to solve the discrete-time nonlinear filtering task in a Bayesian framework. Particle filtering methods use a set of weighted samples (particles) drawn from a proposal distribution to recursively approximate the filtering distribution in time. Unlike EKFs, these algorithms do not rely on linearization of the

dynamic and measurement models and do not require the Gaussian assumption on the predictive and posterior distributions. As a result, particle filters are widely applicable in many challenging nonlinear state-estimation tasks, where the EKF performs poorly. The approximation of the state posterior from a particle filter is statistically consistent [68], i.e., it converges to the true posterior distribution as the number of particles approaches infinity.

Sequential Importance Sampling (SIS)

At each time step k, particle filters aim to approximate the filtering distribution of the hidden state in a recursive manner. The joint posterior of the entire state trajectory $p(x_{0:k}|z_{1:k})$ can be factorized as:

$$\begin{aligned} p(x_{0:k}|z_{1:k}) &\propto p(x_{0:k}, z_{1:k})\,, \\ &= p(z_{1:k-1})p(x_{0:k-1}|z_{1:k-1})p(x_k|x_{k-1})p(z_k|x_k)\,, \\ &\propto p(x_{0:k-1}|z_{1:k-1})p(x_k|x_{k-1})p(z_k|x_k)\,. \end{aligned} \tag{2.36}$$

Suppose the joint posterior distribution of the previous time step $k-1$ is approximated by a set of N_p weighted particles $\{\omega_{k-1}^i, x_{0:k-1}^i\}_{i=1}^{N_p}$ as follows:

$$p(x_{0:k-1}|z_{1:k-1}) \approx \sum_{i=1}^{N_p} \omega_{k-1}^i \delta_{x_{0:k-1}^i}(x_{0:k-1})\,. \tag{2.37}$$

Here, $x_{0:k-1}^i$ denotes the trajectory of the i-th particle and $\omega_{k-1}^i \geqslant 0$ is the importance weight associated with it. The importance weights are assumed to be normalized, i.e., $\sum_{i=1}^{N_p} \omega_{k-1}^i = 1$. Suppose the particles $\{x_{0:k}^i\}_{i=1}^{N_p}$ are sampled from a proposal distribution $q(x_{0:k}|z_{1:k})$. The (unnormalized) importance weights $\{\omega_k^i\}_{i=1}^{N_p}$ are defined as:

$$\omega_k^i \propto \frac{p(x_{0:k}^i|z_{1:k})}{q(x_{0:k}^i|z_{1:k})}\,. \tag{2.38}$$

Assume that the proposal distribution $q(x_{0:k}|z_{1:k})$ can be factorized as follows:

$$\begin{aligned} q(x_{0:k}|z_{1:k}) &= q(x_{0:k-1}|z_{1:k})q(x_k|x_{1:k-1}, z_{1:k}) \,, \\ &= q(x_{0:k-1}|z_{1:k-1})q(x_k|x_{k-1}, z_k) \,, \end{aligned} \tag{2.39}$$

which allows recursive sampling of the state-trajectory and ensures that the computational complexity of the particle filtering algorithm is fixed at each time step. Since $x^i_{0:k-1} \sim q(x_{0:k-1}|z_{1:k-1})$, $x^i_{0:k}$ can be drawn from $q(x_{0:k}|z_{1:k})$ by first sampling $x^i_k \sim q(x_k|x^i_{k-1}, z_k)$ and appending x^i_k to the past trajectory $x^i_{0:k-1}$. Using eqs. (2.36), (2.37), (2.38), and (2.39), the recursive update of importance weights is given by:

$$\begin{aligned} \omega^i_k &\propto \frac{p(x^i_{0:k-1}|z_{1:k-1})p(x^i_k|x^i_{k-1})p(z_k|x^i_k)}{q(x^i_{0:k-1}|z_{1:k-1})q(x^i_k|x^i_{k-1}, z_k)} \,, \\ &\propto \omega^i_{k-1}\frac{p(x^i_k|x^i_{k-1})p(z_k|x^i_k)}{q(x^i_k|x^i_{k-1}, z_k)} \,. \end{aligned} \tag{2.40}$$

The weights are subsequently normalized so that $\sum_{i=1}^{N_p}\omega^i_k = 1$ and the joint posterior distribution of the state at the current time step k is approximated as:

$$p(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N_p} \omega^i_k \delta_{x^i_{0:k}}(x_{0:k}) \,. \tag{2.41}$$

In the filtering task, we are only interested in approximating the marginal posterior distribution $p(x_k|z_{1:k})$, which can be obtained by discarding the past trajectory of each particle as follows:

$$p(x_k|z_{1:k}) \approx \sum_{i=1}^{N_p} \omega^i_k \delta_{x^i_k}(x_k) \,. \tag{2.42}$$

From eq. (2.40), we observe that for the filtering task, only $\{w^i_{k-1}, x^i_{k-1}\}_{i=1}^{N_p}$ need to be stored from the previous time step $k-1$ instead of the full trajectories $\{x^i_{0:k-1}\}_{i=1}^{N_p}$ of the particles. This ensures that the memory requirement does not grow with time.

Resampling

A major drawback of the SIS approach is that the variance of the importance weights increases rapidly with time [69]. In other words, most of the ω_k^is become negligible when k is moderately high and the Monte Carlo approximation of the joint posterior in eq. (2.41) is effectively determined by only a few particles. In the literature, this phenomenon is referred to as weight degeneracy [23, 70] and it results in a poor representation of the posterior distribution. When the state dimension is high and/or when measurements are highly informative, designing a proposal distribution which is well matched to the target posterior becomes computationally challenging, and a larger discrepancy between these two distributions causes high variance of the importance weights.

In order to combat weight degeneracy to some extent, many practical particle filters perform a resampling of the particles after the importance sampling. This operation is termed resampling as it refers to sampling from an approximated distribution, which itself has been obtained by sampling [69]. There are several approaches for resampling in the literature such as multinomial resampling [71], stratified resampling [72], residual resampling [73], and systematic resampling [73, 74]. Detailed studies of the theoretical properties, computational complexity and empirical performance of various resampling schemes can be found in [75–78]. Intuitively, resampling discards particles with negligible weights and makes several copies of the particles with high weights in a probabilistic manner. This allows the particle filter to focus on those regions of the state-space where the posterior is high. A particle filter that uses resampling of the particles is often termed a *Sequential Importance Sampling with Resampling* (SISR) method.

Although in most cases the resampling step improves the estimation performance of particle filters with fixed number of particles, the variances of the SISR algorithm based estimators are greater than the variances of SIS algorithm based estimators for the same proposal distribution [79]. A standard practice in the literature is to assess the weight degeneracy at each time step by computing the *effective sample size* (ESS) [80, 81] using the normalized importance weights as follows:

$$\mathrm{ESS} = \frac{1}{\sum_{i=1}^{N_p} (\omega_k^i)^2} \,. \tag{2.43}$$

From eq. (2.43), we note that $1 \leqslant \text{ESS} \leqslant N_p$. If $\omega_k^i = \frac{1}{N_p}$ for $i = 1, \cdots, N_p$, the ESS becomes N_p. This refers to the case when the proposal distribution matches the target distribution exactly so that importance sampling becomes equivalent to traditional Monte Carlo sampling. On the other hand, if $\omega_k^i = 1$ for only one particle and $\omega_k^i = 0$ for all other particles, ESS takes its lowest value, which corresponds to the most severe degeneracy of the importance weights. Instead of every time step, in most particle filters, resampling is performed only if the ESS is lower than some acceptable threshold (e.g., 50% of N_p). Pseudocode for a generic SISR is provided in Algorithm 2.2.

Algorithm 2.2 A generic SISR algorithm

1: Initialization: Draw $\{x_0^i\}_{i=1}^{N_p}$ from the initial probability density $p(x_0)$. Set $\omega_0^i = \frac{1}{N_p}$ for $i = 1, \cdots, N_p$. Estimate $\hat{x}_0 = \frac{1}{N_p} \sum_{i=1}^{N_p} x_0^i$.
2: **for** $k = 1$ to K **do**
3: Sample particles from the proposal distribution $x_k^i \sim q(x_k | x_{k-1}^i, z_k)$, for $i = 1, \cdots, N_p$.
4: Compute $\tilde{\omega}_k^i = \omega_{k-1}^i \frac{p(x_k^i | x_{k-1}^i) p(z_k | x_k^i)}{q(x_k^i | x_{k-1}^i, z_k)}$ for $i = 1, \cdots, N_p$.
5: Normalize $\omega_k^i = \frac{\tilde{\omega}_k^i}{\sum_{j=1}^{N_p} \tilde{\omega}_k^j}$ for $i = 1, \cdots, N_p$.
6: Estimate $\hat{x}_k = \sum_{i=1}^{N_p} \omega_k^i x_k^i$.
7: **if** ESS $<$ *threshold* **then**
8: Resample $\{\omega_k^i, x_k^i\}_{i=1}^{N_p}$ to obtain $\{\frac{1}{N_p}, x_k^i\}_{i=1}^{N_p}$.
9: **end if**
10: **end for**

Although there is a wide variety of particle filters, they mostly differ from each other in the design of the proposal distribution $q(x_k | x_{k-1}, z_k)$. The *bootstrap particle filter* (BPF) [13] uses the dynamic model to sample the particles, i.e., $q(x_k | x_{k-1}, z_k) = p(x_k | x_{k-1})$. As a result, the importance weight update step in eq. (2.40) only requires the computation of the measurement likelihood at the sampled particles.

$$\begin{aligned} \omega_k^i &\propto \omega_{k-1}^i \frac{p(x_k^i | x_{k-1}^i) p(z_k | x_k^i)}{p(x_k^i | x_{k-1}^i)}, \\ &= \omega_{k-1}^i p(z_k | x_k^i). \end{aligned} \tag{2.44}$$

The implementation of the BPF algorithm is straightforward. However, it can be highly

inefficient if the state dimension is high and/or the measurements are highly informative (in other words, the likelihood is highly localized in certain regions of the state-space). In these cases, the weight degeneracy exhibited by the BPF is severe [70], even if a large number of particles is employed. In order to mitigate the weight degeneracy of the particles, more sophisticated particle filtering methods incorporate the measurement z_k in the construction of their proposals. Although several avenues have been explored for better design of the proposal distribution, high-dimensional particle filtering remains extremely challenging and computationally expensive for many conventional SMC approaches [14, 82, 83].

If the proposal q is chosen such that $q(x_k|x_{k-1}, z_k) = p(x_k|x_{k-1}, z_k)$ is satisfied, then the importance weights in eq. (2.40) do not depend on the sampled particles $\{x_k^i\}_{i=1}^{N_p}$.

$$\begin{aligned}
\omega_k^i &\propto \omega_{k-1}^i \frac{p(x_k^i|x_{k-1}^i)p(z_k|x_k^i)}{p(x_k^i|x_{k-1}^i, z_k)}\,, \\
&= \omega_{k-1}^i \frac{p(x_k^i, z_k|x_{k-1}^i)}{p(x_k^i|x_{k-1}^i, z_k)}\,, \\
&= \omega_{k-1}^i \frac{p(z_k|x_{k-1}^i)p(x_k^i|x_{k-1}^i, z_k)}{p(x_k^i|x_{k-1}^i, z_k)}\,, \\
&= \omega_{k-1}^i p(z_k|x_{k-1}^i)\,. \qquad (2.45)
\end{aligned}$$

This proposal distribution minimizes the variance of the importance weights over different realizations of $\{x_k^i\}_{i=1}^{N_p}$ and is referred to as the 'optimal' proposal distribution in the literature [69, 84]. Although the use of this proposal is desirable, its implementation is impossible in most cases as sampling from $p(x_k|x_{k-1}, z_k)$ cannot be carried out and the importance weight update step in eq. (2.45) requires evaluation of an integral

$$p(z_k|x_{k-1}) = \int p(z_k|x_k)p(x_k|x_{k-1})\,dx_k\,, \qquad (2.46)$$

which cannot be computed in closed form in a general nonlinear, non-Gaussian HMM.

A prominent research direction in the particle filtering literature aims to approximate the optimal proposal distribution using various techniques. A locally linear-Gaussian approximation of the true HMM is employed in [84], whereas [85, 86] consider optimization based approaches to minimize the discrepancy between the target and the proposal distributions. The *auxiliary particle filter* (APF) [87] constructs a measurement-driven

proposal by resampling the particles at time step $k-1$ according to weights that depend on the measurement at time step k. As a result, the particles with higher predictive likelihood are sampled more often than the others, which improves the approximation of the posterior. The unscented transform is used for designing an approximation of the optimal proposal in the Unscented Particle Filter (UPF) [88]. The Rao-Blackwellised particle filter [89] marginalizes some states analytically in order to reduce the variance of the Monte Carlo estimates. The marginal particle filter [90] considers the marginal posterior of the hidden state as the target distribution and thus avoids performing importance sampling in a state-space whose dimension grows with time k. Computational techniques for reducing the complexity of such an approach from $\mathcal{O}(N_p^2)$ to $\mathcal{O}(N_p \log N_p)$ are also proposed. While these filters are statistically consistent, the equivalent weights particle filter [91, 92] sacrifices the statistical consistency of the conventional SISR approach for ensuring that a large number of particles always have substantial weight. This is extremely beneficial for combating weight degeneracy, particularly in high-dimensional scenarios.

Factorization or partitioning of the state-space is another avenue for improving the performance of the particle filters in high dimensions. The *multiple particle filtering* [16, 17] approaches are constituted of several particle filters, run in parallel on low-dimensional subspaces that partition the high-dimensional state-space. The low-dimensional filters need to share their estimates amongst each other for their weight updates. The block particle filter [19] relies on partitioning the state-space into several independent blocks; the measurement space can also be partitioned into corresponding blocks. The filtering distribution of the state variables in one block is updated using only the corresponding block of measurements. However, the performance of this algorithm is heavily dependent on the partitioning/blocking process, since it leads to a bias, which is difficult to characterize. The space-time particle filters [18, 93] use factorization of the conditional posterior to guarantee statistical consistency of the Monte Carlo estimates as the number of particle grows. Although these algorithms are promising, they rely on the posterior having a special structure, which limits their applicability.

Another class of particle filters, which deviates from the generic SISR approach in Algorithm 2.2, uses MCMC algorithms for improving particle diversity. The *resample-move* particle filters [94] perform MCMC moves after the resampling step to improve the quality of the approximate posterior distribution. Similar ideas are adopted in drift homotopy

particle filters [95–97]. These modify the state-dynamics to facilitate the MCMC moves. Although these approaches often show better performance because of the incorporation of MCMC algorithms, the number of MCMC moves required to diversify the particles can be considerable in high-dimensional settings, which results in high computational complexity.

2.1.6 Particle Flow

In the last decade, a new class of Monte Carlo-based filters [20, 21, 24, 98–118] has emerged that achieves impressive performance in high-dimensional filtering. Suppose the posterior at time $k-1$ is approximated by a set of N_p unweighted particles $\{x_{k-1}^i\}_{i=1}^{N_p}$. Propagating the particles through the dynamic model yields $\{\tilde{x}_k^i\}_{i=1}^{N_p}$, which represent the predictive posterior $p(x_k|z_{1:k-1})$ at time k. Particle flow methods gradually migrate these particles $\{\tilde{x}_k^i\}_{i=1}^{N_p}$ towards the correct regions of state-space so that they approximate the posterior distribution $p(x_k|z_{1:k})$ at time k, when the flow is complete. The 'flow' of the particles is typically specified through a partial differential equation (PDE). The weight degeneracy issue associated with a particle filter is absent in particle flow algorithms since the importance sampling using a proposal distribution and the resampling of the particles are eliminated in the latter class of techniques.

A particle flow can be modelled by a background stochastic process η_λ in a pseudo-time interval $\lambda \in [0, 1]$, such that the distribution of η_0 is the predictive distribution $p(x_k|z_{1:k-1})$ and the distribution of η_1 is the posterior distribution $p(x_k|z_{1:k}) = \dfrac{p(x_k|z_{1:k-1})p(z_k|x_k)}{p(z_k|z_{1:k-1})}$. The time index k is temporarily omitted to simplify notation, because the particle flow only concerns migration of particles within a single time step. η_λ^i denotes the stochastic process's i-th realization, and we initialize the flow by setting $\eta_0^i = \tilde{x}_k^i$ for $i = 1, \cdots, N_p$.

In [20, 21], the trajectory of η_λ^i follows an ordinary differential equation (ODE) with zero diffusion resulting in deterministic flows of particles:

$$\frac{d\eta_\lambda^i}{d\lambda} = \varphi(\eta_\lambda^i, \lambda) \ , \tag{2.47}$$

where, $\varphi : \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_x}$ is governed by the Fokker-Planck equation and additional flow constraints [21]. Eq. (2.47) can lead to a variety of particle flow filters. An analytically

tractable solution exists if the predictive distribution is Gaussian and the measurement model is linear with additive Gaussian noise. The exact flow [21] for the linear Gaussian model (specified by eq. (2.4), (2.5), and (2.6)) is:

$$\varphi(\eta_\lambda^i, \lambda) = \frac{d\eta_\lambda^i}{d\lambda} = A(\lambda)\eta_\lambda^i + b(\lambda) \ , \tag{2.48}$$

where,

$$A(\lambda) = -\frac{1}{2}\bar{P}H^T(\lambda H\bar{P}H^T + R)^{-1}H \ , \tag{2.49}$$

$$b(\lambda) = (I + 2\lambda A(\lambda))[(I + \lambda A(\lambda))\bar{P}H^T R^{-1}z + A(\lambda)\bar{\mu}] \ . \tag{2.50}$$

Here, $\bar{\mu}$ and $\bar{P}$ are the mean and the covariance matrix of the predictive posterior distribution, respectively. z denotes the new observation and H is the measurement matrix, i.e., $h(x_k) = Hx_k$. R represents the covariance matrix of the measurement noise. Both $\bar{P}$ and R are assumed to be positive definite matrices. We refer to this method as the exact Daum-Huang (EDH) filter, and its implementation is described in [103]. If the measurement model is nonlinear, a linearization of h is performed at $\bar{\eta}_\lambda = \frac{1}{N_p}\sum_{i=1}^{N_p}\eta_\lambda^i$ and the observation z is corrected for the linearization error e. We define,

$$H(\lambda) = \left.\frac{\partial h(\eta)}{\partial \eta}\right|_{\eta=\bar{\eta}_\lambda} \ , \tag{2.51}$$

$$e(\lambda) = h(\bar{\eta}_\lambda) - H(\lambda)\bar{\eta}_\lambda \, . \tag{2.52}$$

This $H(\lambda)$ is used in place of H and z is replaced by $(z - e(\lambda))$ for computing the EDH flow in eq. (2.49) and (2.50) in a nonlinear setting. The extended Kalman filter (EKF) [65] or the unscented Kalman filter (UKF) [119] is run in parallel to estimate the covariance matrix of the predictive posterior distribution.

Discretized pseudo-time integration is usually employed to approximate the solution to the ODE in eq. (2.47). Suppose that a sequence of discrete steps are taken at N_λ positions $[\lambda_1, \lambda_2, \cdots, \lambda_{N_\lambda}]$, where $0 = \lambda_0 < \lambda_1 < \cdots < \lambda_{N_\lambda} = 1$. The step size $\epsilon_p = \lambda_p - \lambda_{p-1}$ for $p = 1, \cdots, N_\lambda$ can be possibly varying, and we require that $\sum_{p=1}^{N_\lambda}\epsilon_p = \lambda_{N_\lambda} - \lambda_0 = 1$. The linearization of $H(\lambda_p)$ subsequently used to compute $A(\lambda_p)$ and $b(\lambda_p)$ is performed at $\bar{\eta}_{\lambda_{p-1}}$.

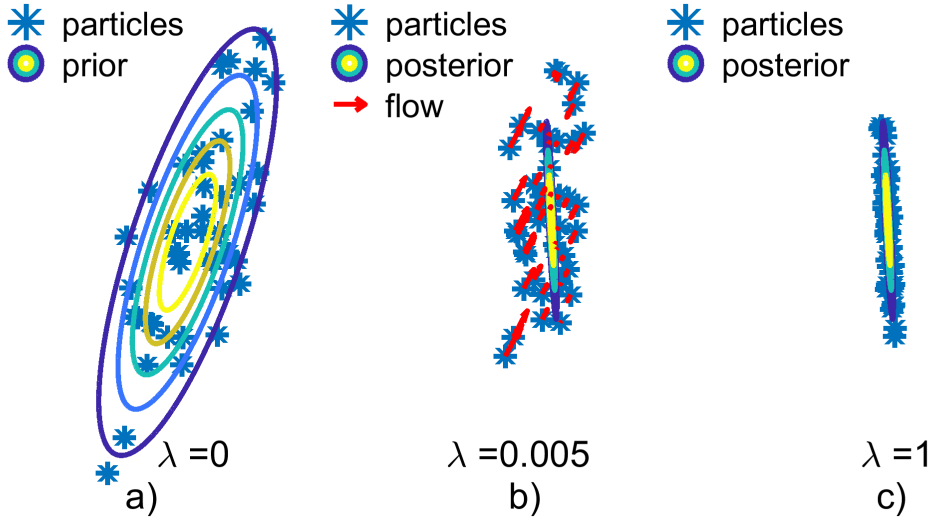


Figure 2.1: Migration of particles from a 2-d Gaussian prior to a 2-d Gaussian posterior distribution. a) The particles/samples (asterisk) from the prior distribution; b) The contours of the posterior distribution and the direction of flow for the particles at an intermediate step; c) The particles after the flow, approximately distributed according to the posterior distribution.

The Euler update rule approximates the integration of the EDH flow of η_λ^i between λ_{p-1} and λ_p as

$$\begin{aligned} \eta_{\lambda_p}^i &= f_{\lambda_p}(\eta_{\lambda_{p-1}}^i) \\ &= \eta_{\lambda_{p-1}}^i + \epsilon_p \left(A(\lambda_p)\eta_{\lambda_{p-1}}^i + b(\lambda_p) \right) . \end{aligned} \tag{2.53}$$

Figure 2.1 demonstrates the migration of the particles from the prior to the posterior distribution according to the EDH flow for a Gaussian predictive distribution and a linear-Gaussian measurement model. The pseudocode for the EDH particle flow is summarized in Algorithm 2.3.

If the measurement function h is highly nonlinear, then the common linearization at $\bar{\eta}_{\lambda_{p-1}}$ for all particles results in large estimation error. A computationally intensive variation of EDH, that linearizes the system and updates the drift term for each individual particle, is proposed in [105]. This approach is referred to as the localized exact Daum-Huang (LEDH)

Algorithm 2.3 EDH particle flow.

Initialization: Draw $\{x_0^i\}_{i=1}^{N_p}$ from the initial probability density $\mathcal{N}(\mu_0, P_0)$. Set $\hat{x}_0 = \frac{1}{N_p}\sum_{i=1}^{N_p} x_0^i$. Set $\lambda_0 = 0$.
for $k = 1$ to K **do**
  Apply EKF/UKF prediction: $\{\hat{x}_{k-1}, P_{k-1}\} \to \{m_{k|k-1}, \bar{P}_k\}$.
  Propagate particles through dynamic model $\eta_0^i = g_k(x_{k-1}^i) + v_k$, for $i = 1, \cdots, N_p$.
  Compute $\bar{\eta}_0 = \frac{1}{N_p}\sum_{i=1}^{N_p} \eta_0^i$, set $\bar{\mu} = \bar{\eta}_0$.
  for $p = 1, \cdots, N_\lambda$ **do**
    Set $\lambda_p = \lambda_{p-1} + \epsilon_p$.
    Calculate $A(\lambda_p)$ and $b(\lambda_p)$ from (2.49) and (2.50) with linearization of h_k performed at $\bar{\eta}_{\lambda_{p-1}}$, and with $z = z_k - e(\lambda_p)$, $\bar{P} = \bar{P}_k$, and $R = R_k$.
    Migrate particles: $\eta_{\lambda_p}^i = \eta_{\lambda_{p-1}}^i + \epsilon_p \left(A(\lambda_p)\eta_{\lambda_{p-1}}^i + b(\lambda_p)\right)$ for $i = 1, \cdots, N_p$.
    Compute $\bar{\eta}_{\lambda_p} = \frac{1}{N_p}\sum_{i=1}^{N_p} \eta_{\lambda_p}^i$.
  end for
  Set $x_k^i = \eta_1^i$ for $i = 1, \cdots, N_p$.
  Apply EKF/UKF update: $\{m_{k|k-1}, \bar{P}_k\} \to \{m_{k|k}, P_k\}$.
  Estimate $\hat{x}_k = \bar{\eta}_1$.
  (Optional) Redraw particles $\{x_k^i\}_{i=1}^{N_p} \sim \mathcal{N}(\hat{x}_k, P_k)$.
end for

filter and is shown to outperform the EDH filter significantly for nonlinear models. For the i-th particle, the drift term of LEDH is

$$\varphi(\eta_\lambda^i, \lambda) = A^i(\lambda)\eta_\lambda^i + b^i(\lambda) \ , \tag{2.54}$$

where,

$$A^i(\lambda) = -\frac{1}{2}\bar{P}H^i(\lambda)^T(\lambda H^i(\lambda)\bar{P}H^i(\lambda)^T + R)^{-1}H^i(\lambda) \ , \tag{2.55}$$

$$b^i(\lambda) = (I + 2\lambda A^i(\lambda))[(I + \lambda A^i(\lambda))\bar{P}H^i(\lambda)^T R^{-1}(z - e^i(\lambda)) + A^i(\lambda)\bar{\mu}] \ , \tag{2.56}$$

$$H^i(\lambda) = \left.\frac{\partial h(\eta)}{\partial \eta}\right|_{\eta=\eta_\lambda^i} \ , \tag{2.57}$$

$$e^i(\lambda) = h(\eta_\lambda^i) - H^i(\lambda)\eta_\lambda^i \, . \tag{2.58}$$

The implementation of the LEDH particle flow is detailed in Algorithm 2.4.

Algorithm 2.4 LEDH particle flow.

Initialization: Draw $\{x_0^i\}_{i=1}^{N_p}$ from the initial probability density $\mathcal{N}(\mu_0, P_0)$. Set $\hat{x}_0 = \frac{1}{N_p}\sum_{i=1}^{N_p} x_0^i$. Set $\lambda_0 = 0$.
for $k = 1$ to K **do**
  Apply EKF/UKF prediction: $\{\hat{x}_{k-1}, P_{k-1}\} \rightarrow \{m_{k|k-1}, \bar{P}_k\}$.
  Propagate particles through dynamic model $\eta_0^i = g_k(x_{k-1}^i) + v_k$, for $i = 1, \cdots, N_p$.
  Compute $\bar{\eta}_0 = \frac{1}{N_p}\sum_{i=1}^{N_p} \eta_0^i$, set $\bar{\mu} = \bar{\eta}_0$.
  for $p = 1, \cdots, N_\lambda$ **do**
    Set $\lambda_p = \lambda_{p-1} + \epsilon_p$.
    for $i = 1, \cdots, N_p$ **do**
      Calculate $A^i(\lambda_p)$ and $b^i(\lambda_p)$ from (2.55) and (2.56) with linearization of h_k performed at $\eta_{\lambda_{p-1}}^i$, and with $z = z_k$, $\bar{P} = \bar{P}_k$ and $R = R_k$.
      Migrate particles: $\eta_{\lambda_p}^i = \eta_{\lambda_{p-1}}^i + \epsilon_p \left(A^i(\lambda_p)\eta_{\lambda_{p-1}}^i + b^i(\lambda_p)\right)$.
    end for
  end for
  Set $x_k^i = \eta_1^i$ for $i = 1, \cdots, N_p$.
  Apply EKF/UKF update: $\{m_{k|k-1}, \bar{P}_k\} \rightarrow \{m_{k|k}, P_k\}$.
  Estimate $\hat{x}_k = \frac{1}{N_p}\sum_{i=1}^{N_p} x_k^i$.
  (Optional) Redraw particles $\{x_k^i\}_{i=1}^{N_p} \sim \mathcal{N}(\hat{x}_k, P_k)$.
end for

The EDH [21, 103] and the LEDH [105] are constructed as zero-diffusion deterministic flow. However, there are several stochastic particle flow approaches [24, 106, 114, 116–118], where a non-zero diffusion term is present in the flow equations. In these cases, the trajectory of the particles follows a stochastic differential equation (SDE). Although the stochastic particle flow approaches sometimes attain better particle diversity compared to their deterministic counterparts, this comes at a cost of increased computational complexity in most cases. There are several other types of particle flow approaches, such as the small curvature flow filter [104], the Coulomb's law flow filter [102], and the renormalization group flow filter [107]. However, the implementation of these filters is often challenging and computationally demanding. Although the particle flow approaches are governed by a continuous differential equation, their practical implementation requires approximation using discrete steps. In many cases, the underlying model assumptions required for deriving various particle flow equations are not satisfied for the true *state-space model* (SSM). These approximation errors and model mismatches cause the particles to deviate

from the true posterior distribution, which can potentially lead to poor performance of particle flow techniques in practical scenarios.

2.1.7 Particle Flow Particle Filter

As discussed in the previous section, the migrated particles after the particle flow process are not exactly distributed according to the posterior because of the discretization errors made while numerically solving eq. (2.47), and the mismatch of modelling assumptions between a general HMM and a linear Gaussian setup (which was assumed in deriving eq. (2.49) and (2.50)). Instead η_1^i can be viewed as being drawn from a proposal distribution $q(\eta_1^i|x_{k-1}^i, z_k)$, which is possibly well matched to the posterior, because of the flow procedure. If the LEDH flow parameters, $(A^i(\lambda_p), b^i(\lambda_p))$ are computed based on linearization of the measurement function h at an auxiliary particle location $\bar{\eta}_{\lambda_p}^i$, starting from $\bar{\eta}_0^i = g_k(x_{k-1}^i, 0)$, then under a mild smoothness condition on h and with small enough step sizes ϵ_p, the discretized particle flow process introduces a deterministic, invertible mapping $\eta_1^i = T^i(\eta_0^i; x_{k-1}^i, z_k)$, as shown in [26]. This property enables efficient evaluation of the importance density:

$$q(\eta_1^i|x_{k-1}^i, z_k) = \frac{p(\eta_0^i|x_{k-1}^i)}{|\det(\dot{T}^i(\eta_0^i; x_{k-1}^i, z_k))|}\,, \tag{2.59}$$

where $\dot{T}^i(\cdot) \in \mathbb{R}^{d\times d}$ is the Jacobian matrix of the mapping function $T^i(\cdot)$ with respect to η_0^i and $|\cdot|$ denotes the absolute value. The determinant of $\dot{T}^i(\cdot)$ is given as:

$$\det(\dot{T}^i(\eta_0^i; x_{k-1}^i, z_k)) = \prod_{p=1}^{N_\lambda} \det(I + \epsilon_p A^i(\lambda_p))\,. \tag{2.60}$$

In [26], a *particle flow particle filter* (PFPF) is proposed which uses the proposal in eq. (2.59). The resulting PFPF (LEDH) algorithm is highly effective in approximating high-dimensional posterior distributions and it has lower run-time compared to several other high-dimensional particle filtering approaches. Its pseudocode is summarized in Algorithm 2.5. A computationally efficient version of the PFPF algorithm based on the EDH flow which uses a common invertible mapping for all particles is also described in [26].

There are several other approaches which use particle flow or optimal transport [120] inside

Algorithm 2.5 PFPF (LEDH)

Initialization: Draw $\{x_0^i\}_{i=1}^{N_p}$ from the initial probability density $p(x_0)$. Set $\omega_0^i = \frac{1}{N_p}$ for $i = 1, \cdots, N_p$. Estimate $\hat{x}_0 = \frac{1}{N_p}\sum_{i=1}^{N_p} x_0^i$. Set $\{P_0^i\}_{i=1}^{N_p}$ to be the covariance of $p(x_0)$. Set $\lambda_0 = 0$.
for $k = 1$ to K **do**
  for $i = 1, \cdots, N_p$ **do**
    Apply EKF/UKF prediction, $\{x_{k-1}^i, P_{k-1}^i\} \to \{m_{k|k-1}^i, \bar{P}_k^i\}$.
    Calculate $\bar{\eta}_0^i = g_k(x_{k-1}^i, 0)$, set $\bar{\mu} = \bar{\eta}_0^i$.
    Propagate particle $\eta_0^i = g_k(x_{k-1}^i, v_k)$.
    Set $\theta^i = 1$.
    for $p = 1, \cdots, N_\lambda$ **do**
      Set $\lambda_p = \lambda_{p-1} + \epsilon_p$.
      Calculate $A^i(\lambda_p)$ and $b^i(\lambda_p)$ from eq. (2.55) and (2.56) with linearization of h_k performed at $\bar{\eta}_{\lambda_{p-1}}^i$ and with $z = z_k$, $\bar{\mu} = \bar{\eta}_0^i$, and $\bar{P} = \bar{P}_k^i$.
      Migrate auxiliary particle: $\bar{\eta}_{\lambda_p}^i = \bar{\eta}_{\lambda_{p-1}}^i + \epsilon_p \left(A^i(\lambda_p)\bar{\eta}_{\lambda_{p-1}}^i + b^i(\lambda_p)\right)$
      Migrate particle: $\eta_{\lambda_p}^i = \eta_{\lambda_{p-1}}^i + \epsilon_p \left(A^i(\lambda_p)\eta_{\lambda_{p-1}}^i + b^i(\lambda_p)\right)$
      $\theta^i = \theta^i |det(I + \epsilon_p A^i(\lambda_p))|$
    end for
    Set $x_k^i = \eta_1^i$
    Calculate importance weights: $\tilde{\omega}_k^i = \omega_{k-1}^i \dfrac{p(x_k^i|x_{k-1}^i)p(z_k|x_k^i)}{p(\eta_0^i|x_{k-1}^i)/\theta^i}$
  end for
  for $i = 1, \cdots, N_p$ **do**
    Normalize $\omega_k^i = \tilde{\omega}_k^i / \sum_{s=1}^{N_p} \tilde{\omega}_k^s$
    Apply EKF/UKF update, $\{m_{k|k-1}^i, \bar{P}_k^i\} \to \{m_{k|k}^i, P_k^i\}$
  end for
  Estimate $\hat{x}_k = \sum_{i=1}^{N_p} \omega_k^i x_k^i$
  (Optional) resample particles : $\{x_k^i, P_k^i, \omega_k^i\}_{i=1}^{N_p}$ to obtain $\{x_k^i, P_k^i, \dfrac{1}{N_p}\}_{i=1}^{N_p}$
end for

a particle filtering framework. The *Guided Sequential Monte Carlo* (GSMC) [22] algorithm constructs a transport map based on Ensemble Kalman filter (EnKF) [121] for designing an efficient proposal distribution for a particle filter. Although, this approach is effective, the computational burden of the EnKF is substantial. The Gibbs flow [25] approach solves the Liouville equation to identify the transport map from the prior to the posterior distribution. Numerical implementation for a d_x-dimensional state involves computation of d_x coupled one-dimensional conditional flows, which does not scale favorably with d_x. Although the Gibbs flow does not require any distributional assumptions about the prior and the posterior, it is not computationally tractable in most cases.

The proposal distribution in the stochastic particle flow filter [24] is formed based on the stationary solution of Fokker-Planck equation with non-zero diffusion. Since the numerical solution is approximate, importance sampling is required to account for the mismatch between the target and the proposal distributions. Although this approach shows impressive performance in high dimensions, evaluation of the diffusion matrix in each flow step results in a significantly high computational complexity.

Bunch et al. employ approximate Gaussian flow to design a proposal distribution for a particle filter in the *Gaussian particle flow importance sampling* (GPFIS) [23] algorithm. The underlying particle flow is modelled by a stochastic differential equation, which can be solved analytically for a linear-Gaussian HMM. This solution is used to approximate the optimal flow for nonlinear models. In this approach, the proposal distribution cannot be evaluated in closed form in general, instead the authors derive and numerically solve a differential equation for the importance weights. It is also shown that if the step size to integrate the particle flow is decreased to approach zero, this algorithm ensures that the particles after the flow are properly weighted so that they can approximate the current step posterior distribution. The overall algorithm can be interpreted as an effective approximation of the optimal proposal distribution within the SMC framework. Despite its impressive performance in many high-dimensional settings, a major disadvantage of this algorithm is its computational burden, which is exceptionally high, mostly due to computation of the square roots of matrices and solution of the Sylvester equation.

2.1.8 Sequential Markov Chain Monte Carlo

A promising research direction to combat weight degeneracy in high-dimensional particle filtering is to use *Markov chain Monte Carlo* (MCMC) methods to improve the diversity of samples. Sequential Markov chain Monte Carlo (SMCMC) [15, 27, 61, 122] methods were proposed as an effective alternative to particle filters, especially for challenging high-dimensional state-estimation problems. MCMC is often considered as the most effective method for sampling from a high-dimensional distribution [123]. More effective MCMC techniques in high-dimensional spaces use Hamiltonian or Langevin dynamics to construct efficient proposals [124–126]. Although effective, these techniques cannot be directly used in sequential inference tasks, as MCMC typically targets a static distribution. To use MCMC to diversify samples in a sequential setting, the resample-move particle filter incorporates MCMC methods by performing MCMC moves after the resampling step of the particle filte [94]. Unfortunately, the resampling step can lead to degeneracy, so many MCMC moves may be required to diversify particles.

Sequential Markov chain Monte Carlo (SMCMC) methods use MCMC techniques to sample directly from the approximate target distribution. A unifying framework of the various SMCMC methods was provided in [15]. In [122], sampling directly from the filtering distribution is targeted. This approach is computationally expensive since it requires the numerical integration of the predictive density at each time step. Instead, sampling from the joint state distribution is proposed in [15, 27, 61]. In the algorithms presented in [15, 27, 61], a fixed number of samples is used to approximate the empirical posterior distribution for each time step. By contrast, in the *sequentially interacting MCMC* (SIMCMC) framework described in [127], one can continue to generate interacting non-Markovian samples of the entire state sequence to improve the empirical approximation of joint posterior distribution successively. The resulting samples are asymptotically distributed according to the joint posterior distribution. The fundamental difference between the SMCMC and the SIMCMC techniques is that the SMCMC algorithm consists of sequential implementation of static MCMC schemes (justifying the name 'sequential MCMC'), whereas this interpretation does not hold for the SIMCMC algorithm. Hence the analysis of SIMCMC in [127] cannot be applied to SMCMC (and vice-versa) and SMCMC cannot be expressed as a special case of SIMCMC. From a

practical viewpoint, if a fixed number of particles is to be used, the effect of error in approximating the posterior distribution at the previous time step might be severe for the SIMCMC algorithm and limit its applicability in high-dimensional online filtering problems compared to advanced SMC or SMCMC techniques.

As shown in eq. (2.36) in Section 2.1.5, the joint posterior distribution $\pi_k(x_{0:k}) = p(x_{0:k}|z_{1:k})$ can be computed pointwise up to a normalizing constant in a recursive manner:

$$\begin{aligned}\pi_k(x_{0:k}) = p(x_{0:k}|z_{1:k}) &\propto p(x_{0:k}, z_{1:k})\,, \\ &= p(x_k|x_{k-1})p(z_k|x_k)p(x_{0:k-1}|z_{1:k-1})p(z_{1:k-1})\,, \\ &\propto p(x_k|x_{k-1})p(z_k|x_k)\pi_{k-1}(x_{0:k-1})\,. \end{aligned} \tag{2.61}$$

As $\pi_{k-1}(x_{0:k-1})$ is not analytically tractable, it is impossible to sample from it in a general HMM. In all SMCMC methods, the distribution is replaced by its empirical approximation in (2.61), which leads to an approximation of π_k as follows:

$$\breve{\pi}_k(x_{0:k}) \propto p(x_k|x_{k-1})p(z_k|x_k)\widehat{\pi}_{k-1}(x_{0:k-1})\,, \tag{2.62}$$

where,

$$\widehat{\pi}_{k-1}(x_{0:k-1}) = \frac{1}{N_p}\sum_{j=N_b+1}^{N_b+N_p} \delta_{x^j_{k-1,0:k-1}}(x_{0:k-1})\,. \tag{2.63}$$

Here $\delta_a(\cdot)$ is the Dirac delta function centred at a, N_b is the number of samples discarded during a burn-in period, and N_p is the number of retained MCMC samples. $\{x^j_{k-1,0:k-1}\}_{j=N_b+1}^{N_b+N_p}$ are the N_p samples obtained from the Markov chain at time $k-1$, whose stationary distribution is $\breve{\pi}_{k-1}(x_{0:k-1})$. At time step k, $N_b + N_p$ iterations of the Metropolis-Hastings (MH) algorithm [5] with proposal $q_k(\cdot)$ are executed to generate samples $\{x^j_{k,0:k}\}_{j=N_b+1}^{N_b+N_p}$ from the invariant distribution $\breve{\pi}_k(x_{0:k})$, and $\pi_k(x_{0:k})$ is approximated as:

$$\widehat{\pi}_k(x_{0:k}) = \frac{1}{N_p}\sum_{j=N_b+1}^{N_b+N_p} \delta_{x^j_{k,0:k}}(x_{0:k})\,. \tag{2.64}$$

The purpose of the joint draw of $\breve{\pi}_k(x_{0:k})$ is to avoid numerical integration of the predictive density when the target distribution is $p(x_k|z_{1:k})$ [27]. Note that if we are only interested in approximating the marginal posterior distribution $p(x_k|z_{1:k})$, only $\{x^j_{k-1,k-1}\}_{j=N_b+1}^{N_b+N_p}$ needs to be stored instead of the full past state trajectories $\{x^j_{k-1,0:k-1}\}_{j=N_b+1}^{N_b+N_p}$. The Metropolis-Hastings (MH) algorithm used within SMCMC to generate one sample is summarized in Algorithm 2.6.

Algorithm 2.6 MH Kernel in SMCMC [15].
Input: $x^{i-1}_{k,0:k}$.
Output: $x^i_{k,0:k}$.

1: Propose $x^{*(i)}_{k,0:k} \sim q_k(x_{0:k}|x^{i-1}_{k,0:k})$.
2: Compute the MH acceptance probability $\rho = \min\left(1, \frac{\breve{\pi}_k(x^{*(i)}_{k,0:k})}{q_k(x^{*(i)}_{k,0:k}|x^{i-1}_{k,0:k})} \frac{q_k(x^{i-1}_{k,0:k}|x^{*(i)}_{k,0:k})}{\breve{\pi}_k(x^{i-1}_{k,0:k})}\right)$.
3: Accept $x^i_{k,0:k} = x^{*(i)}_{k,0:k}$ with probability ρ, otherwise set $x^i_{k,0:k} = x^{i-1}_{k,0:k}$.

Composite MH Kernel in SMCMC

Different choices of the MCMC kernel for high-dimensional SMCMC are discussed in [15]. In most SMCMC algorithms, an independent MH kernel is adopted [15], i.e., $q_k(x_{0:k-1}|x^{i-1}_{k,0:k}) = q_k(x_{0:k})$, meaning that the proposal is independent of the state of the Markov chain at the previous iteration. The ideal choice is the optimal independent MH kernel, i.e., $q_k(x_{0:k}) = \breve{\pi}_k(x_{0:k})$ such that the MH acceptance rate in line 2 of Algorithm 2.6 is always 1. However, as shown in [15], it is impossible to sample from the proposal density $p(x_k|z_k, x_{k-1})$ or evaluate $p(z_k|x_{k-1}) = \int p(z_k|x_k)p(x_k|x_{k-1})dx_k$ in most cases. It is difficult to identify an effective approximation to the optimal independent MH kernel. The choice of independent MH kernel using the prior as the proposal can lead to very low acceptance rates if the state dimension is very high or the measurements are highly informative.

Septier et al. propose the use of a composite MH kernel [15, 27], which is constituted of a joint proposal $q_{k,1}$ (to update $x_{0:k}$) followed by two individual state variable refinements using proposals $q_{k,2}$ (to update $x_{0:k-1}$) and $q_{k,3}$ (to update x_k), based on the *Metropolis within Gibbs* approach, within a single MCMC iteration. The composite kernel approach is summarized in Algorithm 2.7.

Any of the MCMC kernels mentioned before can be used in the joint draw step of a composite

Algorithm 2.7 Composite MH Kernels in a unifying framework of SMCMC [15, 27].
Input: $x_{k,0:k}^{(i-1)}$.
Output: $x_{k,0:k}^{(i)}$.

Joint draw of $x_{k,0:k}^{i}$:

1: Propose $x_{k,0:k}^{*(i)} \sim q_{k,1}(x_{0:k}|x_{k,0:k}^{i-1})$.

2: Compute the MH acceptance probability $\rho_1 = \min\left(1, \frac{\breve{\pi}_k(x_{k,0:k}^{*(i)})}{q_{k,1}(x_{k,0:k}^{*(i)}|x_{k,0:k}^{i-1})} \frac{q_{k,1}(x_{k,0:k}^{i-1}|x_{k,0:k}^{*(i)})}{\breve{\pi}_k(x_{k,0:k}^{i-1})}\right)$.

3: Accept $x_{k,0:k}^{i} = x_{k,0:k}^{*(i)}$ with probability ρ_1, otherwise set $x_{k,0:k}^{i} = x_{k,0:k}^{i-1}$.

Individual refinement of $x_{k,0:k-1}^{i}$:

4: Propose $x_{k,0:k-1}^{*(i)} \sim q_{k,2}(x_{0:k-1}|x_{k,0:k}^{i})$.

5: Compute the MH acceptance probability $\rho_2 = \min\left(1, \frac{\breve{\pi}_k(x_{k,0:k-1}^{*(i)}, x_{k,k}^{i})}{q_{k,2}(x_{k,0:k-1}^{*(i)}|x_{k,0:k}^{i})} \frac{q_{k,2}(x_{k,0:k-1}^{i}|x_{k,0:k-1}^{*(i)}, x_{k,k}^{i})}{\breve{\pi}_k(x_{k,0:k}^{i})}\right)$.

6: Accept $x_{k,0:k-1}^{i} = x_{k,0:k-1}^{*(i)}$ with probability ρ_2.

Individual refinement of $x_{k,k}^{i}$:

7: Propose $x_{k,k}^{*(i)} \sim q_{k,3}(x_k|x_{k,0:k}^{i})$.

8: Compute the MH acceptance probability $\rho_3 = \min\left(1, \frac{\breve{\pi}_k(x_{k,0:k-1}^{i}, x_{k,k}^{*(i)})}{q_{k,3}(x_{k,k}^{*(i)}|x_{k,0:k}^{i})} \frac{q_{k,3}(x_{k,k}^{i}|x_{k,0:k-1}^{i}, x_{k,k}^{*(i)})}{\breve{\pi}_k(x_{k,0:k}^{i})}\right)$.

9: Accept $x_{k,k}^{i} = x_{k,k}^{*(i)}$ with probability ρ_3.

---

kernel. For example, the independent MH kernel based on the prior as the proposal is used in the joint draw step of the implementation of the sequential manifold Hamiltonian Monte Carlo (SmHMC) algorithm in [15] as follows:

1. Sample $x_{k,0:k-1}^{*(i)} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$.
2. Sample $x_{k,k}^{*(i)} \sim p(x_k|x_{k,k-1}^{*(i)})$.

Using eq. (2.62), the MH acceptance rate of the joint draw step in Algorithm 2.7 can be simplified as :

$$\begin{aligned}\rho_1 &= \min\left(1, \frac{\breve{\pi}_k(x_{k,0:k}^{*(i)})\widehat{\pi}_{k-1}(x_{k,0:k-1}^{i-1})p(x_{k,k}^{i-1}|x_{k,k-1}^{i-1})}{\widehat{\pi}_{k-1}(x_{k,0:k-1}^{*(i)})p(x_{k,k}^{*(i)}|x_{k,k-1}^{*(i)})\breve{\pi}_k(x_{k,0:k}^{i-1})}\right), \\ &= \min(1, \frac{p(z_k|x_{k,k}^{*(i)})}{p(z_k|x_{k,k}^{i-1})}) \ . \end{aligned} \tag{2.65}$$

For individual refinement of $x_{k,0:k-1}^{i}$, [15] uses the independent proposal $q_{k,2} = \widehat{\pi}_{k-1}$, which

leads to the following simplification of MH acceptance rate in Line 5 of Algorithm 2.6, using eq. (2.62).

$$\begin{aligned}\rho_2 &= \min\Big(1, \frac{\breve{\pi}_k(x^{*(i)}_{k,0:k-1}, x^i_{k,k})\widehat{\pi}_{k-1}(x^i_{k,0:k-1})}{\widehat{\pi}_{k-1}(x^{*(i)}_{k,0:k-1})\breve{\pi}_k(x^i_{k,0:k})}\Big), \\ &= \min\Big(1, \frac{p(x^i_{k,k}|x^{*(i)}_{k,k-1})}{p(x^i_{k,k}|x^i_{k,k-1})}\Big). \end{aligned} \tag{2.66}$$

The aim of the refinement steps is to explore the neighborhood of samples generated in the joint draw step. For the MCMC kernel of the individual refinement step of x_k, Langevin diffusion [128] or Hamiltonian dynamics [125] have been proposed to more efficiently traverse a high-dimensional space [15]. The manifold Hamiltonian Monte Carlo (mHMC) [129] kernel $q_{k,3}(\cdot)$ of the individual refinement step of the SmHMC algorithm efficiently samples from the target filtering distribution, making the SmHMC algorithm one of the most effective algorithms for filtering in high-dimensional spaces.

In [55, 61], the capability of particle flow to migrate samples into high posterior density regions is utilized to construct a composite Metropolis-Hasting (MH) kernel that significantly increases the acceptance rate of the joint draw. Here, we briefly review the SMCMC methods with particle flow, since these approaches are closely related to the methodology presented in the next chapter. In Appendix A, we derive some convergence results for the SMCMC algorithms.

2.1.9 SMCMC with Invertible Particle Flow

In order to construct MH kernels based on invertible particle flow, a new formulation of the invertible mapping with particle flow is considered.

Invertible Mapping with Particle Flow

The particle flow particle filters (PFPFs) [26] construct invertible particle flows in a pseudo-time interval $\lambda \in [0, 1]$ in order to gradually move particles drawn from the prior distribution into regions where the posterior density is high.

Using the Euler update rule specified in eq. (2.53) recursively over $p = N_\lambda, N_\lambda - 1, \cdots, 2, 1$, the invertible mapping for the PF-PF (EDH) can be expressed as:

$$\begin{aligned}\eta_1^i =& f_{\lambda_{N_\lambda}}(f_{\lambda_{N_\lambda-1}}(\cdots f_{\lambda_1}(\eta_0^i))\,,\\ =&(I+\epsilon_{N_\lambda}A(\lambda_{N_\lambda}))\eta^i_{\lambda_{N_\lambda-1}}+\epsilon_{N_\lambda}b(\lambda_{N_\lambda})\,,\\ =&\cdots\\ =&C\eta_0^i+D\ ,\end{aligned} \tag{2.67}$$

where

$$C=\prod_{p=1}^{N_\lambda}\left(I+\epsilon_{N_\lambda+1-p}A(\lambda_{N_\lambda+1-p})\right)\ , \tag{2.68}$$

and

$$D=\epsilon_{N_\lambda}b(\lambda_{N_\lambda})+\sum_{m=1}^{N_\lambda-1}\left(\left[\prod_{p=1}^{N_\lambda-m}\left(I+\epsilon_{N_\lambda+1-p}A(\lambda_{N_\lambda+1-p})\right)\right]\epsilon_p b(\lambda_p)\right)\ . \tag{2.69}$$

In [26], an auxiliary flow is constructed such that the deterministic flow parameters (C, D) do not depend on η_0^i. It is shown that the equivalent mapping is invertible with sufficiently small ϵ_p, so the matrix C is invertible. The procedure to obtain C and D by performing an auxiliary particle flow based on successive linearization of the measurement function h at intermediate auxiliary particle location $\tilde{\eta}_{\lambda_{p-1}}$, starting from $\tilde{\eta}_0$ is summarized in Algorithm 2.8. Since, the migration of the particles in eq. (2.67) is equivalent to applying an affine transform with deterministic parameters, the proposal density can be evaluated as follows:

$$q(\eta_1^i|x_{k-1}^i,z_k)=\frac{p(\eta_0^i|x_{k-1}^i)}{|\det(C)|}\,. \tag{2.70}$$

Similarly, for the PF-PF (LEDH), the invertible mapping can be expressed as

$$\eta_1^i=C^i\eta_0^i+D^i\ , \tag{2.71}$$

where different flow parameters (C^i, D^i) are computed for each particle. The proposal density

Algorithm 2.8 Computation of the flow parameters (C, D).
Input: $\tilde{\eta}_0, P, R, z, \bar{\mu}$.
Output: (C, D).

1: Initialize: $C = I, D = \mathbf{0}$.
2: **for** $p = 1, \cdots, N_\lambda$ **do**
3: Set $\lambda_p = \lambda_{p-1} + \epsilon_p$.
4: Calculate $A(\lambda_p)$ and $b(\lambda_p)$ from eq. (2.55) and (2.56) with linearization of h performed at $\tilde{\eta}_{\lambda_{p-1}}$.
5: Migrate particle $\tilde{\eta}_{\lambda_p} = \tilde{\eta}_{\lambda_{p-1}} + \epsilon_p \left(A(\lambda_p)\tilde{\eta}_{\lambda_{p-1}} + b(\lambda_p)\right)$.
6: Set $C = \left(I + \epsilon_p A(\lambda_p)\right)C$.
7: Set $D = \left(I + \epsilon_p A(\lambda_p)\right)D + \epsilon_p b(\lambda_p)$.
8: **end for**

becomes

$$q(\eta_1^i | x_{k-1}^i, z_k) = \frac{p(\eta_0^i | x_{k-1}^i)}{|\det(C^i)|} \ . \tag{2.72}$$

SmHMC with LEDH

The composite MH kernel using the invertible mapping established by the auxiliary LEDH flow is presented in Algorithm 2.9. In the i-th MCMC iteration at time step k, $x_{k,0:k-1}^{*(i)} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$ is sampled from the approximate joint posterior distribution at time $k-1$. Then, an auxiliary LEDH particle flow starting from $\bar{\eta}_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, 0)$ using Algorithm 2.8 is performed to obtain the auxiliary LEDH flow parameters $(C^{*(i)}, D^{*(i)})$, and this flow is applied to the propagated particle $\eta_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, v_k)$. Thus the proposed particle is generated as: $x_{k,k}^i = \eta_1^{*(i)} = C^{*(i)}\eta_0^{*(i)} + D^{*(i)}$.

For this proposal, the acceptance rate of the joint draw in Algorithm 2.9 can be derived using eq. (2.62) and (2.72):

Algorithm 2.9 Composite MH Kernels constructed with the manifold Hamiltonian Monte Carlo kernel and the invertible particle flow with LEDH, at the i-th MCMC iteration of k-th time step.
Input: $x_{k,0:k}^{i-1}, \eta_0^{i-1}, C^{i-1}$.
Output: $x_{k,0:k}^{i}, \eta_0^{i}, C^{i}$.

Joint draw of $x_{k,0:k}^{i}$:

1: Draw $x_{k,0:k-1}^{*(i)} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$.
2: Sample $\eta_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, v_k)$.
3: Calculate $\bar{\eta}_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, 0)$.
4: Perform invertible particle flow (Algorithm 2.8) to compute $(C^{*(i)}, D^{*(i)})$ by starting the auxiliary particle flow from $\bar{\eta}_0^{*(i)}$.
5: Calculate $x_{k,k}^{*(i)} = C^{*(i)}\eta_0^{*(i)} + D^{*(i)}$.
6: Compute the MH acceptance probability $\rho_1 = \min\left(1, \frac{p(x_{k,k}^{*(i)}|x_{k,k-1}^{*(i)})p(z_k|x_{k,k}^{*(i)})|\det(C^{*(i)})|p(\eta_0^{i-1}|x_{k,k-1}^{i-1})}{p(\eta_0^{*(i)}|x_{k,k-1}^{*(i)})p(x_{k,k}^{i-1}|x_{k,k-1}^{i-1})p(z_k|x_{k,k}^{i-1})|\det(C^{i-1})|}\right)$.
7: Accept $x_{k,0:k}^{i} = x_{k,0:k}^{*(i)}$, $\eta_0^i = \eta_0^{*(i)}$, $C^i = C^{*(i)}$ and $D^i = D^{*(i)}$ with probability ρ_1. Otherwise set $x_{k,0:k}^{i} = x_{k,0:k}^{i-1}$, $\eta_0^i = \eta_0^{i-1}$, $C^i = C^{i-1}$ and $D^i = D^{i-1}$.
8: Perform individual refinements of $x_{k,0:k}^{i}$ using Algorithm 2.10.
9: Calculate $\eta_0^i = (C^i)^{-1}(x_{k,k}^i - D^i)$.

$$\begin{aligned}\rho_1 &= \min\left(1, \frac{\breve{\pi}_k(x_{k,0:k}^{*(i)})\widehat{\pi}_{k-1}(x_{k,0:k-1}^{i-1})q(x_{k,k}^{i-1}|x_{k-1}^{i-1}, z_k)}{\widehat{\pi}_{k-1}(x_{k,0:k-1}^{*(i)})q(x_{k,k}^{*(i)}|x_{k-1}^{*(i)}, z_k)\breve{\pi}_k(x_{k,0:k}^{i-1})}\right), \\ &= \min\left(1, \frac{p(x_{k,k}^{*(i)}|x_{k,k-1}^{*(i)})p(z_k|x_{k,k}^{*(i)})|\det(C^{*(i)})|p(\eta_0^{i-1}|x_{k,k-1}^{i-1})}{p(\eta_0^{*(i)}|x_{k,k-1}^{*(i)})p(x_{k,k}^{i-1}|x_{k,k-1}^{i-1})p(z_k|x_{k,k}^{i-1})|\det(C^{i-1})|}\right). \end{aligned} \tag{2.73}$$

When evaluating eq. (2.73) in Line 5 of Algorithm 2.9, the values of $x_{k,k}^{i-1}$, η_0^{i-1} and C^{i-1} are needed. Since $x_{k,k}^{i-1}$ may be generated by the manifold Hamiltonian Monte Carlo kernel $q_{k,3}(\cdot)$ as shown in Algorithm 2.10, the corresponding η_0^{i-1} is not available through Lines 2 and 6 of Algorithm 2.9. This can be resolved using the invertible mapping property of the invertible particle flow. As C^{i-1} is invertible, η_0^{i-1} can be calculated from $x_{k,k}^{i-1}$ by solving eq. (2.71):

$$\eta_0^{i-1} = (C^{i-1})^{-1}(x_{k,k}^{i-1} - D^{i-1}) . \tag{2.74}$$

Algorithm 2.10 Individual refinement steps of composite MH Kernels constructed with the manifold Hamiltonian Monte Carlo kernel, at the i-th MCMC iteration of k-th time step.
Input: $x^i_{k,0:k}$.
Output: $x^i_{k,0:k}$.

Individual refinement of $x^i_{k,0:k-1}$:

1: Draw $x^{*(i)}_{k,0:k-1} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$.

2: Compute the MH acceptance probability $\rho_2 = \min\left(1, \frac{p(x^i_{k,k}|x^{*(i)}_{k,k-1})}{p(x^i_{k,k}|x^i_{k,k-1})}\right)$.

3: Accept $x^i_{k,0:k-1} = x^{*(i)}_{k,0:k-1}$ with probability ρ_2.

Individual refinement of $x^i_{k,k}$:

4: Propose $x^{*(i)}_{k,k} \sim q_{k,3}(x_k|x^i_{k,k-1:k}, z_k)$ using the manifold Hamiltonian Monte Carlo kernel.

5: Compute the MH acceptance probability $\rho_3 = \min\left(1, \frac{p(x^{*(i)}_{k,k}|x^i_{k,k-1})p(z_k|x^{*(i)}_{k,k})}{q_{k,3}(x^{*(i)}_{k,k}|x^i_{k,k-1:k},z_k)} \frac{q_{k,3}(x^i_{k,k}|x^i_{k,k-1},x^{*(i)}_{k,k},z_k)}{p(x^i_{k,k}|x^i_{k,k-1})p(z_k|x^i_{k,k})}\right)$.

6: Accept $x^i_{k,k} = x^{*(i)}_{k,k}$ with probability ρ_3.

SmHMC with EDH

Calculation of individual flow parameters at every MCMC iteration in Algorithm 2.9 can be computationally expensive. Similar to the spirit of the PF-PF (EDH) [26], common flow parameters C and D can be computed only once using an auxiliary state variable derived from the samples, and these flow parameters can be applied for all MCMC iterations. The resulting procedure is described in Algorithm 2.11. The flow parameters C and D are calculated only once in the initialization of each time step k, using Algorithm 2.12.

The calculation of the acceptance rate in the joint draw step can be further simplified compared to eq. (2.73) as the same mapping of the flow is applied to each particle. The candidate particle $x^{*(i)}_{k,k}$ and the particle $x^{i-1}_{k,k}$ share the same value of C in their proposal densities in eq. (2.70). Thus, for the SmHMC algorithm with the EDH flow, the acceptance rate of the joint draw is:

$$\rho_1 = \min\left(1, \frac{p(x^{*(i)}_{k,k}|x^{*(i)}_{k,k-1})p(z_k|x^{*(i)}_{k,k})p(\eta_0^{i-1}|x^{i-1}_{k,k-1})}{p(\eta_0^{*(i)}|x^{*(i)}_{k,k-1})p(x^{i-1}_{k,k}|x^{i-1}_{k,k-1})p(z_k|x^{i-1}_{k,k})}\right) \quad . \tag{2.75}$$

Algorithm 2.11 Composite MH Kernels constructed with the manifold Hamiltonian Monte Carlo kernel and the invertible particle flow with EDH, at the i-th MCMC iteration of k-th time step. C and D were pre-computed using Algorithm 2.12.
Input: $x_{k,0:k}^{i-1}, \eta_0^{i-1}, C, D$.
Output: $x_{k,0:k}^{i}, \eta_0^{i}$.

Joint draw of $x_{k,0:k}^{i}$:

1: Draw $x_{k,0:k-1}^{*(i)} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$.
2: Sample $\eta_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, v_k)$.
3: Calculate $x_{k,k}^{*(i)} = C\eta_0^{*(i)} + D$.
4: Compute the MH acceptance probability $\rho_1 = \min\left(1, \frac{p(x_{k,k}^{*(i)}|x_{k,k-1}^{*(i)})p(z_k|x_{k,k}^{*(i)})p(\eta_0^{i-1}|x_{k,k-1}^{i-1})}{p(\eta_0^{*(i)}|x_{k,k-1}^{*(i)})p(x_{k,k}^{i-1}|x_{k,k-1}^{i-1})p(z_k|x_{k,k}^{i-1})}\right)$.
5: Accept $x_{k,0:k}^{i} = x_{k,0:k}^{*(i)}$, $\eta_0^i = \eta_0^{*(i)}$ with probability ρ_1.
Otherwise set $x_{k,0:k}^{i} = x_{k,0:k}^{i-1}$, $\eta_0^i = \eta_0^{i-1}$.
6: Individual refinements of $x_{k,0:k}^{i}$ using Algorithm 2.10.
7: Calculate $\eta_0^i = C^{-1}(x_{k,k}^i - D)$.

Algorithm 2.12 The flow parameter calculation for SmHMC (EDH).

1: Draw $x_{k,0:k-1}^{0} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$.
2: Sample $\eta_0^0 = g_k(x_{k,k-1}^0, v_k)$.
3: Calculate $\bar{\eta}_0 = g_k(\bar{x}_{k-1,k-1}, 0)$.
4: Perform invertible particle flow (Algorithm 2.8) to compute (C, D) by starting the auxiliary particle flow from $\bar{\eta}_0$.

2.2 Graph Neural Networks and Graph Generative Models

In this section, we provide a brief review of learning apparatus adapted to graph structured data. Beginning with a gentle introduction of earlier *graph neural network* (GNN) approaches and a general framework that describes many existing GNN models, we focus primarily on the *graph convolutional networks* (GCNs), since they are adopted in our work. We discuss the GNN models that are closely related to the methodology presented in this thesis subsequently. We also present some material on data-driven graph inference algorithms, parametric random graph models, and machine learning based graph generative approaches, since some of these techniques are employed in the implementation of the Bayesian methodology proposed in

this thesis.

2.2.1 Graph Neural Networks

Most traditional supervised and unsupervised machine learning algorithms deal with classification, regression, and representation learning of *independent and identically distributed* (i.i.d.) data instances. If there is a regular structure, such as the regular grid present in images, *convolutional neural networks* (CNNs) [130, 131] constitute a highly successful learning approach by exploiting local translational invariance of such data to extract rich features. Similarly, *recurrent neural networks* (RNNs) [132, 133] are applied widely in addressing learning tasks concerning sequential data. However, the ubiquity of graph structured data in many application areas, such as, wireless networks, social networks, biological networks, poses several challenges to the conventional machine learning algorithms. The i.i.d. assumption for different entities does not hold for data residing on the vertices of a graph, since the graph topology might represent correlation or causal relationships among the data instances, depending on the strength and directionality of the edges. The irregular, non-Euclidean structure prohibits straightforward application of existing CNNs to analyze data on graphs. In addition, since there is no notion of a natural ordering of the nodes of a graph in most cases, recurrent architectures are not ideal for these settings. As a result, there has been an extensive research focus on building neural architectures for efficient processing of graph data by addressing these issues.

Early work led to the development of several GNN models [32–34, 134]. These approaches rely on recurrent architectures for recursive processing and message passing for propagation of information across the graph. Training can often take a long time and the required time scales undesirably with respect to the number of nodes in the graph, restricting the use of such models to smaller graphs. An approach for improving the scalability of GNNs by partitioning the graph into several subgraphs was proposed in [135]. This technique alternates between propagating information among nodes in local subgraphs and propagating information among the subgraphs.

Although there is a great variety of GNNs, most of them are constructed based on the same high level idea. They strive to learn a deeper representation (embedding) for each node in the graph using a standard neural network architectures such as a *multi-layer perceptron*

(MLP) and then account for the graph structure by allowing the node embeddings at each layer to be influenced by the embeddings of the neighboring nodes from the previous layer. Formally, a graph $\mathcal{G}$ uses its vertices to represent entities, whose pairwise relationships are encoded by the edges. Thus, we define $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where, $\mathcal{V} = \{1, 2, \cdots, N\}$ denotes the set of N nodes and $\mathcal{E}$ is the set of (directed) edges. We assume that the ordering of the nodes is arbitrary but fixed, i.e., each node is represented by a unique index $i \in \mathcal{V}$. An ordered pair $(i, j) \in \mathcal{E}$ implies that there is an edge from node i to node j. We do not permit self-loops for any node, so $(i, i) \notin \mathcal{E}, \forall i \in \mathcal{V}$. For an undirected graph, the links are bidirectional, i.e, $(i, j) \in \mathcal{E}$ iff $(j, i) \in \mathcal{E}$. For any node i, we define the set of its neighbors as $\mathcal{N}_i = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$. Suppose $\boldsymbol{x}_i$ and $\boldsymbol{x}_{ij}$ denote the feature vector associated with node i and link (i, j) respectively. We denote the representation of node i and edge (i, j) at ℓ-th layer by $\boldsymbol{h}_i^{(\ell)}$ and $\boldsymbol{h}_{ij}^{(\ell)}$.

The *message passing neural network* (MPNN) [35] framework performs the following recursive operation at each layer for updating the node representations.

$$\boldsymbol{h}_{ij}^{(\ell+1)} = f_{\text{message}}^{(\ell)}(\boldsymbol{h}_i^{(\ell)}, \boldsymbol{h}_j^{(\ell)}, \boldsymbol{x}_{ij}), \forall (i, j) \in \mathcal{E}, \tag{2.76}$$

$$\boldsymbol{a}_i^{(\ell+1)} = f_{\text{aggregate}}^{(\ell)}(\{\boldsymbol{h}_{ij}^{(\ell+1)} : j \in \mathcal{N}_i\}), \forall i \in \mathcal{V}, \tag{2.77}$$

$$\boldsymbol{h}_i^{(\ell+1)} = f_{\text{combine}}^{(\ell)}(\boldsymbol{h}_i^{(\ell)}, \boldsymbol{a}_i^{(\ell+1)}), \forall i \in \mathcal{V}. \tag{2.78}$$

Here, $f_{\text{message}}^{(\ell)}(\cdot, \cdot, \cdot)$ computes the message from node j to i at ℓ-th layer using a standard neural architecture. $f_{\text{aggregate}}^{(\ell)}(\cdot)$ aggregates the messages from all neighbors of node i. Typical choices for this function include neighborhood sampling followed by concatenation [136] or weighted sum [37, 38]. $f_{\text{combine}}^{(\ell)}$ combines the aggregated message from the neighborhood with $\boldsymbol{h}_i^{(\ell)}$ to compute the representation of node i at the $(\ell + 1)$-th layer. It usually applies learnable weights and a nonlinear activation function. It is shown in [35] that the MPNN model encompasses many existing GNNs, including various graph convolutional architectures [37, 62, 137, 138] and gated GNNs [134]. Successive developments, such as the *graph isomorphism network* (GIN) [139] can also be cast in this framework. However, eqs. (2.76), (2.77), and (2.78) cannot describe all GNNs. For example, the node representations potentially depend on global attributes of the graph in [140], which cannot be represented by these steps.

For addressing node classification or regression problems [37, 38], several message passing

layers are stacked and this overall architecture is trained via backpropagation. Similarly, edge-level tasks, such as link prediction [39, 40], can be carried out by forming the output for each edge using the embeddings of the incident nodes. A graph embedding for graph classification [136, 139] can be obtained using a readout function after the last layer. Recent surveys on various GNN architectures are provided in [141, 142].

2.2.2 Graph Convolutional Networks

Graph convolutional networks (GCNs) have emerged more recently, with the first proposals in [137, 138, 143]. These approaches use spectral graph convolution [144], which we now review briefly. Let $\mathbf{A} \in \mathbb{R}_+^{N \times N}$ denote the adjacency matrix of the graph $\mathcal{G}$, i.e.,

$$\mathbf{A}_{i,j} = \begin{cases} a_{i,j}, & \text{if } (i,j) \in \mathcal{E} \\ 0, & \text{otherwise.} \end{cases} \tag{2.79}$$

Here, we denote the non-negative weight of edge (i, j) by $a_{i,j}$. For the rest of this section, we assume that the graph $\mathcal{G}$ is undirected, i.e. $a_{i,j} = a_{j,i}$. As there are no self-loops in $\mathcal{E}$, $a_{i,i} = 0$, $\forall 1 \leqslant i \leqslant N$. If the graph is unweighted, then $a_{i,j} \in \{0, 1\}$. However, for a weighted graph $a_{i,j} \geqslant 0$. The degree of node i is defined as: $d_i = \sum_{j=1}^{N} a_{i,j}$. The degree matrix $\mathbf{D} \in \mathbb{R}_+^{N \times N}$ is diagonal and positive semidefinite with diagonal entries, $\mathbf{D}_{i,i} = d_i$. So, we can write, $\mathbf{D} = diag(\mathbf{A1})$, where $\mathbf{1}$ is N-dimensional column vector of all ones and $diag(\cdot)$ returns a diagonal matrix of a column vector. The (unnormalized) graph Laplacian, $\widetilde{\mathbf{L}} = \mathbf{D} - \mathbf{A}$, can be noramalized to $\mathbf{L} = \mathbf{D}^{-1/2}\widetilde{\mathbf{L}}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. $\mathbf{L}$ is a real symmetric positive semi-definite matrix, whose complete set of orthonormal eigenvectors $\{\mathbf{u}_i\}_{i=0}^{N-1}$ spans $\mathbb{R}^N$ and are termed the graph Fourier modes [144]. The (ordered) real non-negative eigenvalues $\{\lambda_i\}_{i=0}^{N-1}$, are referred to as the frequencies of the graph $\mathcal{G}$. The Laplacian matrix can be diagonalized as: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where the graph Fourier basis $\mathbf{U} = [\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}] \in \mathbb{R}^{N \times N}$ is constructed by stacking the eigenvectors columnwise and $\mathbf{\Lambda} = diag([\lambda_0, \ldots, \lambda_{N-1}]^T) \in \mathbb{R}_+^{N \times N}$ is the diagonal matrix of the corresponding eigenvalues.

If $\mathbf{x} \in \mathbb{R}^N$ denotes a signal on the vertices of graph $\mathcal{G}$, its Graph Fourier Transform (GFT) is defined as: $\hat{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$. Using the orthonormality of $\mathbf{U}$, the inverse GFT can also be obtained as $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$. Since a suitable translation operator in the vertex domain cannot be

characterized, convolution of two graph signals $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ is defined in the Fourier domain, $\mathbf{x} *_{\mathcal{G}} \mathbf{y} = \mathbf{U}\Big((\mathbf{U}^T\mathbf{x}) \odot (\mathbf{U}^T\mathbf{y})\Big)$, where $\odot$ denotes Hadamard product.

Similarly, spectral convolution of a graph signal $\mathbf{x}$ with a graph filter $\mathbf{g_w} = diag(\mathbf{w}) \in \mathbb{R}^{N \times N}$ to obtain an output $\mathbf{y}$ can be computed as: $\mathbf{y} = \mathbf{U}\mathbf{g_w}\mathbf{U}^T\mathbf{x}$. Here, the entries of the vector of Fourier coefficients, $\mathbf{w} \in \mathbb{R}^N$, are the free parameters of the non-parametric filter $\mathbf{g_w}$.

There are several disadvantages associated with non-parametric filtering. We need to compute $\mathbf{U}$, which has $\mathcal{O}(N^2)$ runtime and the parameter complexity is $\mathcal{O}(N)$, i.e., it grows with the number of nodes. Thus non-parametric filtering based GNN models [143] cannot be scaled to large graphs.

A computationally efficient approach involves expressing the Fourier coefficients of the filter as a function of the eigenvalues of $\mathbf{L}$. For example, if a $K-1$-th order polynomial filter is applied, then we can express the filter as:

$$\mathbf{g}_{\mathbf{w}'}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} w'_k \mathbf{\Lambda}^k \,, \tag{2.80}$$

and the computation of the output $\mathbf{y}$ can be simplified as:

$$\mathbf{y} = \mathbf{U}\mathbf{g}_{\mathbf{w}'}(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x} = \sum_{k=0}^{K-1} w'_k \mathbf{L}^k \mathbf{x} \,. \tag{2.81}$$

This approach has a fixed complexity of $\mathcal{O}(K)$ for learning the new parameters $\mathbf{w}' \in \mathbb{R}^K$, irrespective of how large N is. Moreover, we do not need to compute the Fourier basis $\mathbf{U}$ for filtering. Evaluation of eq. (2.81) scales as $\mathcal{O}(K|\mathcal{E}|)$ using sparse matrix multiplication.

Multi-scale clustering of nodes is combined with spectral convolution in [137] to derive a graph neural network that mimics the CNN in Euclidean data. However, this approach is not scalable to large graphs because of the increased parameter complexity associated with the multi-scale analysis. The model in [138] applies a convolution-like propagation rule on graphs for a graph classification task. However, this approach uses degree-specific weight matrices, which requires extensive computation if the degree distribution has a large variance.

These early models are simplified to obtain a Chebyshev polynomial based spectral filtering

approach in [62]. Defferrard et al. modifies eq. (2.80) slightly to obtain:

$$\mathbf{g_w}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} w_k T_k(\widetilde{\mathbf{\Lambda}})^k \,, \tag{2.82}$$

where $T_k(\cdot)$ is the k-th order Chebyshev polynomial and $\widetilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{max}}\mathbf{\Lambda} - \mathbf{I}$ has eigenvalues in $[-1, 1]$, which improves the numerical stability during model training. If an input $\mathbf{X} \in \mathbb{R}^{N \times d_x}$ has d_x input channels and the output $\mathbf{Y} \in \mathbb{R}^{N \times d_y}$ requires d_y feature maps, then a single layer of the K-th order ChebyNet first applies the filter in eq. (2.82) to the input and then passes the filtered result through a potentially nonlinear activation function $\sigma(\cdot)$ to compute:

$$\mathbf{Y} = \sigma\Big(\sum_{k=0}^{K-1} \widetilde{\mathbf{L}}^k \mathbf{X} \mathbf{W}_k \Big) \,. \tag{2.83}$$

Here $\widetilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}$ and the weight matrices $\{\mathbf{W}_k \in \mathbb{R}^{d_x \times d_y}\}_{k=0}^{K-1}$ are learned through backpropagation. For the design of the popular GCN model [37] of Kipf and Welling, $K = 2$ is used in eq. (2.83) with additional approximations to obtain:

$$\mathbf{Y} = \sigma\Big(\mathbf{A}_{\mathcal{G}} \mathbf{X} \mathbf{W} \Big) \,, \tag{2.84}$$

where $\mathbf{A}_{\mathcal{G}} = (\mathbf{I} + \mathbf{D})^{-1/2}(\mathbf{I} + \mathbf{A})(\mathbf{I} + \mathbf{D})^{-1/2}$ is the normalized, modified adjacency matrix (a self-loop is added at each of the nodes). More sophisticated spectral filters are designed in [145, 146] for improved spatial localization.

While the models discussed so far are based on spectral filtering, several spatial filtering methods or aggregation strategies were adopted in [136, 147]. A general framework for training neural networks on graphs and manifolds was presented by [148] and Monti et al. explain how several of the other methods can be interpreted as special cases. The performance of the graph convolutional neural networks can be improved by incorporating attention mechanism over the neighboring nodes [38], leading to the graph attention network (GAT). Experiments have also demonstrated that gates, edge conditioning, and skip connections can prove beneficial [149–151]. In some problem settings it is also beneficial to consider an ensemble of graphs [152], multiple adjacency matrices [153] or the dual graph [154].

GCN models are mostly employed in a transductive setting, where features of the test set nodes are required to be stored and used for computing the node embeddings during training. Additionally, due to the message passing structure of the GCNs, the receptive field size grows exponentially for each node with respect to the number of layers. Thus, the computational complexity and memory requirement becomes a bottleneck for applying GCNs to large, dense graphs. Several algorithms have been proposed with the goal of improving training efficiency and scalability based on node sampling [155], neighbor sampling [156], layer-wise sampling [157], layer-dependent importance sampling [158], sub-graph sampling [159], and graph clustering [160].

The GCN propagation rule applies low-pass filtering to the node representations [161], i.e., nodes within each others receptive fields tend to have similar representations. As the number of layers increases, this leads to over-smoothing and eventual convergence to indistinguishable representations of nodes [162–164]. Several approaches have been proposed to address this issue including adaptive adjustment of the receptive field [162], characterization of the continuous dynamics of node representations [165], improved propagation schemes using PageRank [166], decoupling of the representation transformation and the message propagation [167], and stochastic regularization [168, 169].

2.2.3 Topology Uncertainty in Graph Neural Networks

In Chapter 4, we propose a Bayesian framework and treat the observed graph as additional data to be used during inference. We published a paper disseminating this research in 2019 [56]. Following the proposal of our general framework to deal with the issue of uncertainty on graphs, several related techniques [41–45] that jointly perform inference of the graph while addressing a learning task such as node classification have been proposed recently. In [41], variational inference is employed to learn the graph structure. This formulation allows consideration of additional data such as features and labels when performing graph inference, but the technique is still tied to a parametric model. The approach in [42] is based on a similar idea; this algorithm forms several k-nearest neighbor graphs using the embedding of each layer of the GCN and subsequently use these graphs in conjunction with the observed graph to estimate a better approximation of the 'true'

topology based on parametric graph modelling. An iterative optimization is adopted for joint optimization of the graph and the GCN weights. In [43], Elinas et al. take a non-parametric approach, but their probabilistic formulation is focused on improving only very noisy graphs. In [44], simultaneous optimization of the graph structure along with the learning task is considered. A contrastive framework is adopted in [45], which combines a generative model similar to the *variational graph autoencoder* (VGAE) [39] to generate another view for the node embeddings. In all of these works, only the node classification task has been explored. Our methodology extends the applicability of the Bayesian framework beyond node classification by incorporating a more flexible non-parametric graph model.

2.2.4 Learning a Graph from Observed Data

Originating primarily from the signal processing community, several algorithms focus exclusively on learning a graph from some observed data [170–172]. In this setting, data entities are associated with different nodes and the connectivity pattern among them is to be inferred. The learned graph might be unweighted [170] or weighted [172]. The central idea followed in many of these techniques is that if two nodes are highly similar, then it is more likely that there is a link (for the unweighted graph case) or a higher edge weight (for the weighted graph case) between them. The notion of this similarity between nodes depends on the assumptions on the observed data, the graph to be estimated, and the relationship between them. Since this task is often unsupervised, i.e., there is no 'true' underlying graph, the evaluation of the estimated topology is not well-defined and depends heavily on the application domain. These methods include statistical approaches which rely on (partial) correlations or coherence measures [173, Chapter 7] among signals, *approximate nearest neighbor* (A-NN) methods [170, 174, 175], Gaussian graphical model based estimation techniques [171, 176, 177], algorithms relying on some signal smoothness criterion [172, 178–180], and diffusion process based methods [181]. A detailed survey of these algorithms is provided in [182]. These approaches differ from our work considerably, since the end goal of the graph learning algorithms is topology inference, whereas our objective is to address node- or edge-level learning tasks for data on graphs.

The correlation based approaches are often informal, requiring tuning of thresholds of ad-

hoc scoring functions. Statistical model based algorithms perform poorly if the underlying assumptions are not satisfied. Assessing the usefulness of inferred graphs in some appropriate sense is difficult in many cases. Many of these methods have $\mathcal{O}(N^2)$ complexity, where N is the number of nodes. As a result, they do not scale well to large graphs. A-NN graph learning [170, 174, 175] has $\mathcal{O}(N \log N)$ complexity, which is more suitable for large scale applications, but the learned graph generally has poor quality compared to the k-NN graph. The algorithms [172, 178, 179], which typically appeal to a smoothness criterion for the graph by minimizing a regularized version of Dirichlet energy [183] (also known as the total variation [184] of the graph signal with respect to the Laplacian matrix), also have $\mathcal{O}(N^2)$ complexity. However, a more recent method in [180] introduces an approximate graph learning algorithm which provides an efficient trade off between runtime and the quality of the solution. We use this approach for the *maximum a posteriori* (MAP) estimation of the non-parametric graph in our Bayesian framework. However, our graph model is tailored to the specific learning task we address.

2.2.5 Parametric Random Graph Models

There is a rich history of research and development of parametric models of random graphs. These models have been designed to generate graphs exhibiting specific characteristics and their theoretical properties have been studied in depth. They can yield samples representative of an observed graph provided that the model is capable of representing the particular graph structure and parameter inference is successful. The Barabasi-Albert model [185], exponential random graphs [186], and exchangeable random graph models [187] fall into this category. Configuration model [188, 189] variants preserve the degree sequence of a graph while changing the connectivity pattern in the random samples. A recent survey of various random graph models is provided in [190].

Detection of community memberships of the nodes of graph structured data has many applications, including analyzing collaboration networks [191], protein interaction networks [192], and social networks [193]. Communities can be loosely defined as sets of nodes which have dense internal connectivity and few external connections [194]. A class of models, termed *stochastic block models* (SBMs), is used extensively in the literature [195–198] for modelling such community structures. The stochastic block model

assumes that each node can only participate in one of the communities and different node pairs having the same community memberships are stochastically equivalent. While this is simple to understand, in real world networks, often a node can be a part of more than one community with varying membership strengths. To address this possibility, the *mixed membership stochastic block model* (MMSBM) is proposed in [199]. Exact inference in this model becomes intractable in large networks and several algorithms for approximate inference have been developed [200, 201]. For the implementation of our Bayesian framework in a node classification setting in Chapter 4, we adopt the MMSBM as the parametric generative model of the observed graph. The overlapping SBMs [202, 203] provide another approach for allowing nodes to belong to multiple communities. Another shortcoming of the SBM is that it cannot accurately model a heavy tailed degree distribution [204]. Such degree distributions are observed in many real world networks. Several modifications to tackle degree heterogeneity within a community have been proposed in various degree-corrected SBMs [196, 204–208].

Despite interpretable modelling of specific graph properties, a disadvantage of using these parametric graph models is that they impose relatively strict structural assumptions in order to maintain tractability. As a result, they often cannot model characteristics like large clustering coefficients [209], small world connectivity and exponential degree distributions [210], observed in many real-world networks. Additionally, most cannot readily take into account node or edge attribute information, and high-dimensional parameter inference can be prohibitively computationally expensive for larger graphs.

2.2.6 Machine Learning Based Graph Models

Various generative graph models have emerged from the machine learning community, incorporating an autoencoder structure. These models are commonly trained to accurately predict the links in the graph, and as a result, tend to fail to reproduce global structural properties of the observed graph [211]. A variational autoencoder model, whose inference distribution is parameterized by a graph convolutional network (GCN), is introduced in [39] to learn node embeddings. The link probabilities are derived from the dot product of the obtained node embeddings. A similar approach is adopted in [212], but the proposed algorithm employs adversarial regularization to learn more robust embeddings. Both of

these models exhibit impressive clustering of node embeddings. Adding a message passing component to the decoder based on intermediately learned graphs is considered in [40]. This leads to improved representations and better link prediction. The strengths of the parametric models and the graph-based learning methods are combined in [213] to propose the *deep generative latent feature relational model* (DGLFRM), which aims to retain the interpretability of the overlapping SBM (OSBM) paired with the flexibility of the graph autoencoder. The incorporation of the OSBM improves the ability of the model to capture block-based community structure. In Chapter 4, we incorporate many of these architectures in the design of the Bayesian versions of these methods using non-parametric graph inference for improved link-prediction.

An alternative approach is to use generative adversarial networks (GANs) as the basis for graph models. Edge probability is modelled through an adversarial framework in the GraphGAN [214] model. The NetGAN model represents the graph as a distribution on random walks [215]. Compared to autoencoder methods, the GAN based methods seem more capable of capturing the structural characteristics of an observed graph. The major disadvantage is that the models are extremely computationally demanding to train and the success of the training can be sensitive to the choice of hyperparameters.

We focus on learning a graph model based on a single observed graph. By contrast, there is a growing body of work that aims to learn graph models that can reproduce graphs that have characteristics similar to a dataset of multiple training graphs. Evaluation is typically done based on the likelihood of sampled graphs and comparing graph characteristics. These approaches can preserve important structural attributes of the graph(s) in the dataset, but the sampled graphs do not retain node identity information. So, they cannot be applied in the node- and edge-oriented learning tasks we focus on. In this category, there have been variational autoencoder approaches [216, 217], GAN-based approaches [218], models based on iterative generation [219], normalizing flow based models [220], and auto-regressive models [221, 222].

2.3 Time-Series Forecasting

This section provides a literature review of relevant time-series forecasting techniques. We discuss statistical forecasting models in Section 2.3.1. Although these approaches allow

for interpretable modelling of the time-series in most cases, as a result of recent advances, neural network-based techniques have started to outperform the statistical approaches for multivariate time-series prediction. We present an overview of these deep learning based algorithms in Section 2.3.2. Recently, powerful multivariate forecasting algorithms that are capable of providing uncertainty characterization have been proposed. We provide a review of these approaches in Section 2.3.3. In some settings, a graph is available that specifies spatial or causal relationships between the time-series. Numerous algorithms have been proposed that combine GNNs with temporal neural network architectures. Algorithms that take into account the graph provide superior forecasts, if the graph is accurate and the indicated relationships have predictive power. These deep learning based spatio-temporal forecasting models are outlined in Section 2.3.4.

Another related area is constituted of various flavours of stochastic recurrent networks [223, 224], which have been introduced for modelling of time-series data in an unsupervised setting. In most cases, variational inference is applied to learn model parameters, although sequential Monte Carlo has also been employed [86, 225, 226]. Many of these methods determine the parameters of sequential Monte Carlo models via optimizing Monte Carlo objectives [225–227]. These approaches are discussed in Section 2.3.5.

2.3.1 Statistical Forecasting Models

Historically, time-series forecasting has been studied by extensive application of various statistical models to fit the time-series data. Some common approaches include univariate forecasting based on *Auto-Regressive* (AR) [46] and *Auto-Regressive Moving Average* (ARMA) [228] models. Relying on a weak stationarity assumption, such models aim for a parsimonious representation of time-series. Generalization of ARMA by introducing first differencing steps results in the *Auto-Regressive Integrated Moving Average* (ARIMA) [47] model, which is capable of handling a non-stationary trend in the mean of the time-series data. However, these approaches are not capable of modelling seasonality. Incorporation of additional seasonal terms in the ARIMA model leads to the *Seasonal ARIMA* (SARIMA) [229] model, which can represent periodic patterns. When used in a multivariate problem, prediction capability of these univariate forecasting techniques is rather limited, since they ignore any correlation between different time-series. For

modelling the relationships among multiple time-series, AR and ARMA models can be generalized to *Vector Auto-Regressive* (VAR) and *Vector Auto-Regressive Moving Average* (VARMA) [230] models. A promising recent advancement in statistical modelling of time-series data is Prophet [231], which is a nonlinear regression approach incorporating various components to capture a trend, multiple seasonalities, and the effects of irregular events, e.g., holidays. Although its impressive performance on small-scale datasets and interpretable modelling makes Prophet a widely used algorithm in many practical tasks, it cannot outperform deep learning based forecasting models on large datasets [48].

If the number of time-series is large compared to the total duration of individual time-series, fitting a VAR model for forecasting becomes challenging. Various types of sparse VAR models, which aim to represent the dependency among individual time-series by using probabilistic graphical models [232] or ℓ_1-type non-smooth regularization [233–235], have been proposed to address such scenarios. If a graph is available in addition to the multivariate time-series, the Graph VARMA [236] model incorporates it by designing suitable graph filters. Kernel-based methods [237, 238] are derived for the cases where a linear model is inadequate to represent the complexity of the time-series data. Although, these approaches are theoretically appealing, the recent deep learning models often achieve significantly better forecasting accuracy for real-world time-series datasets and exhibit better scalability.

2.3.2 Deep Learning Based Point Forecasting Models

Although there is a history of persistent successes of *state-of-the-art* statistical forecasting techniques on many small to medium sized datasets, these models often show poor scalability to a larger number of time-series and rely heavily on hand-crafted features or domain knowledge. These issues potentially hinder the applicability of such models to many practical settings. Recent years have seen significant advances in the development of effective point forecasting models using neural networks. Several types of architectures have been considered, including recurrent network based models [239–245], matrix factorization based approaches [48, 246], and the deep learning approach of [49].

Recurrent neural networks (RNNs) [132, 133, 247–250] are a class of neural network architectures with feedback connections to process sequence data of variable lengths. Recurrent models process the input sequences to recursively update their internal states

(memory), which are subsequently used to form the predictions. This feature makes them a popular choice in diverse application areas including natural language processing [250], text generation [251], audio modelling [252], image generation [253], and time-series forecasting [254]. Despite their widespread use, many primitive RNN architectures cannot learn long-term dependencies in the sequence data well [255], primarily due to gradient vanishing and exploding [256] during training. The most effective RNNs are equipped with gates to control the flow of information to combat this issue. Such gated RNNs include the *long short-term memory* (LSTM) [133] and the *gated recurrent unit* (GRU) [250]. In particular, the LSTM has the capability to adaptively memorize or forget historical information to mitigate gradient vanishing or exploding for processing long sequences. Another drawback of RNN models is that they have limited capacity to represent complex dependencies among multiple variables [245].

Several modifications and innovations have been proposed for using recurrent architectures in time-series forecasting task. Among them, the approach in [239] combines an LSTM with the wavelet transform and a *stacked autoencoder* (SAE) [257]. First, the time-series data is denoised using a wavelet-based decomposition. Subsequently, a SAE is employed to generate high-level representations, which are used as inputs of a LSTM model for forecasting. The *long and short-term time-series network* (LSTNet) [240] approach combines an RNN with a *convolutional neural network* (CNN) [130, 131]. The first layer consists of a CNN without pooling, which strives to learn time-localized patterns. Its output is then fed to a RNN with skip connections to capture long-term tendencies of the time-series data.

The attention mechanism [258] provides an alternate avenue for representing long-term dependencies in time-series data. In addition, it also has the capability to extract meaningful interactions between different time-series, which can be beneficial in multivariate forecasting problems. The *dual-stage attention-based RNN* (DA-RNN) [241] model employs an encoder-decoder architecture for forecasting one step ahead. The encoder is an RNN endowed with an attention module to determine which dimensions of the multivariate time-series are important. The decoder forms the forecast by applying a temporal attention mechanism, which focuses more on relevant encoder states. Although temporal attention is effective in determining which time-lags are important for forecasting, its point-wise nature is not particularly suitable for capturing continuous periodical patterns present in many real-world datasets. The *memory time-series network*

(MTNet) [242] architecture strives to address this issue by designing a long-term memory component. It has three encoders, each of which is constructed by stacking a convolutional layer that extracts time-localized features, an attention mechanism that determines their importance, and a GRU that summarizes the output of the attention module. Two out of the three encoders use long-term historical data as their inputs to build the memory component, which allows the model to remember long-term temporal patterns. The third encoder processes the recent historical data to obtain short-term contexts. Finally, outputs of the memory unit are combined with the short-term contexts to obtain the forecasts.

Another limitation of temporal attention is its inability to incorporate the effect of different importance of input variables in a multivariate setting. In the *multi-variable LSTM* (MV-LSTM) [243] model, Guo and Lin modify the LSTM architecture to accommodate tensorized hidden states such that each element of the hidden state tensor captures information regarding a specific input variable. Successively, they apply a temporal and variable attention mechanism, designed using the hidden states, for forecasting. This specific implementation of the attention mechanism is analogous to the probabilistic mixture of experts model [251], where each variable and lag pair acts as an expert, and the attention mechanism computes the confidence scores for these experts' predictions. A similar motivation is considered for the design of the *adaptive input selection RNN* (AIS-RNN) [244] model, which combines an RNN with an AIS module to incorporate potentially different importance of various input variables and lags. However, the AIS module is built using another RNN, instead of an attention mechanism. Another attention based approach is the hybrid model in [245], which combines a stacked residual LSTM encoder-decoder architecture with a multi-attention module, comprised of temporal and layer-wise attentions. It also incorporates *ensemble empirical mode decomposition* (EEMD) [259] to decompose the data into multiple sub-series, which are subsequently clustered to reduce the computational complexity for obtaining the predictions. However, this architecture cannot be trained end-to-end since the prediction step is separate form the decomposition and clustering operations. Another major drawback of this model is that in spite of the claim of tackling a multivariate forecasting task, this approach essentially focuses on the recent historical data for each individual time-series for forming its predictions.

The *temporal regularized matrix factorization* (TRMF) [246] framework views multivariate time-series modelling as a regularized matrix factorization problem, which supports temporal dependency modelling. The main technical novelty in the TRMF approach stems from the use of an autoregressive model (AR) to describe the time evolution of the low-dimensional temporal coefficients and incorporating it as a regularizer term in the matrix factorization problem. The resulting algorithm naturally addresses the tasks of missing data imputation and forecasting based on the learned AR model of temporal coefficients. Contrary to the graph-based regularization [260] employed in prior temporal matrix factorization models [261, 262], the TRMF model shows improved forecasting performance, since it is capable of capturing negative temporal correlations through the design of its temporal regularizer. Another benefit of this approach is that compared to several statistical forecasting techniques, such as the AR model [46] and linear SSM-based learning [64], TRMF scales more favourably with respect to the number of time-series present in the multivariate data. However, the TRMF method can only represent linear temporal dependencies, which might be inadequate in modelling complex, real-world datasets. Moreover, this approach solely focuses on the global patterns arising form the AR model for forming its predictions. This can result in poor approximations of temporally local behaviours [48].

The DeepGLO [48] model is another related multivariate forecasting approach, which learns both global and local patterns during training and utilizes them for prediction. This is in contrast to univariate models, such as [52–54], which form the forecasts for individual time-series using only the historical evolution of the same time-series, despite being trained on the whole dataset. The hybrid architecture of DeepGLO consists of a global module for capturing common features across all time-series and a local prediction module for learning local individual properties. The global module uses a matrix factorization approach, which is regularized by a temporal convolution [263] operation. The local prediction module is another temporal convolution network, whose predictions are constructed by using the obtained factors from the global module as covariates. Similar to TRMF [246], this approach can scale to thousands of time-series because of the computational efficiency of the underlying matrix factorization component. Empirical results show the efficacy of DeepGLO in handling multivariate data with drastically different scales across individual time-series.

The *neural basis expansion analysis for time-series* (N-BEATS) [49] model is a recently

proposed univariate time-series forecasting method, which demonstrates state-of-the-art prediction accuracy on several large-scale benchmark datasets. Unlike the recurrent models, which are inherently sequential in nature, this approach considers the multi-horizon forecasting as a nonlinear multivariate regression problem. The basic building block of the N-BEATS architecture consists of several fully connected layers, followed by interpretable basis functions to represent local trend and seasonality. Each block has two outputs, termed forecast and backcast. Several blocks are connected hierarchically using doubly-residual links to form a stack. Multiple stacks are used with residual connections to form the overall forecast. This design facilitates interpretable decomposition of the forecasts across stacks; deeper stacks focus on predicting more complex temporal patterns, whereas shallower stacks learn the global features. Since this architecture consists of fully connected layers and residual links, the training time is considerably lower compared to recurrent models.

The deep learning models presented in this section provide point forecasts of the future, as the model parameters are learned by optimizing suitable point forecasting criteria. As a result, these models cannot assess the forecast uncertainty by forming confidence intervals around the mean/median forecasts. Another drawback is that they cannot utilize spatial dependencies between time-series in the form of a graph adjacency for improving their predictions.

2.3.3 Deep Learning Based Probabilistic Forecasting Models

In recent years, there has been substantial research focus on applying deep learning for probabilistic forecasting. For example, the hybrid model combining exponential smoothing with a recurrent network in [254] was the official winner of the M4 competition on forecasting. More sophisticated probabilistic forecasting methods include DeepAR [52], DeepFactors [53], DeepState [54], the Multi-horizon Quantile Recurrent Neural Network (MQRNN) [264], the Gaussian copula process approach of [265], and deep Rao-Blackwellised particle filtering approach of [266]. Normalizing flow [267] has also been combined with temporal neural network architectures [268, 269].

DeepAR [52] is a univariate forecasting algorithm based on an autoregressive recurrent network architecture. The probabilistic forecast is assumed to have a Gaussian distribution

for real valued time-series. The negative binomial density is chosen for modelling count data. The mean and variance of the Gaussian forecast or the mean and shape of the negative binomial forecast are parameterized by applying linear or softplus layers to the output of the recurrent network. The model is trained end to end by minimizing the negative log-likelihood of the forecasts on the training data. Typical implementation of this approach is based on a multi-layer LSTM [133] network, which uses the recent history of the time-series to be predicted along with any additional covariates as its input. Handling the covariates in this way alleviates the need to perform elaborate and dataset specific feature engineering to learn the seasonal behaviour.

DeepFactors [53] is a more general hybrid, univariate forecasting framework which combines a global, non-random, deep learning architecture for capturing efficient representations with a local classical time-series model to handle forecast uncertainty. Typically, one or several RNNs are used to design the global component, whereas the local randomness for different time-series is implemented using additive Gaussian noise, a linear dynamic model, or a Gaussian process. If the forecast likelihood is non-Gaussian, a variational lower bound is maximized for learning the model parameters. The DeepAR architecture can be obtained as a specific instantiation of the DeepFactors framework.

The DeepState [54] approach performs Kalman filtering of the hidden states of a time-varying, linear-Gaussian (SSM) to address univariate probabilistic forecasting. The time-dependent parameters of the SSM are different for each time-series in the dataset, and are learned using a multi-layer recurrent architecture, which uses the additional covariates as its input. Although, the linear-Gaussian modelling adopted in this approach results in improved interpretability for the obtained forecasts, the DeepState model has several disadvantages. A linear-Gaussian SSM might not be able to adequately model complex, rapidly varying temporal patterns, which are present in many practical datasets. Since the parameterization of the underlying time-varying, linear-Gaussian SSM depends on the associated covariates, the forecasting performance can be poor in settings where the covariates are loosely correlated with the time-series of interest or where no meaningful covariate is available. A generalization of the DeepState model is the *deep state-space model for forecasting* (DSSMF) [270] approach, which sacrifices the analytically tractable inference of linear-Gaussian hidden states in order to achieve more effective modelling of the state-transition and emission processes using nonlinear, neural architectures. A

variational inference procedure is adopted for computing the approximate posterior of hidden states.

In the *Multi-Quantile RNN* (MQRNN) [264] model, a multi-quantile loss is minimized during training to learn several quantiles of the multi-horizon forecasts. This approach is an example of the encoder-decoder architecture, which addresses the sequence to sequence [251] prediction task. The encoder is composed of multiple LSTM layers, which summarizes the historical evolution of the time-series to be predicted and the associated covariates. The decoder consists of two *multi-layer perceptrons* (MLPs), one of which, termed the global MLP, uses the the encoder output and the future covariates to form both horizon-specific context vectors for each prediction horizon and a horizon-agnostic context to capture the common information. The quantile predictions are formed by applying the other MLP, which is shared across horizons and uses the two contexts and the corresponding future covariate as its input. This approach has the capability to accommodate both temporal and static covariates, shifting seasonality, known events that cause large spikes, and cold starts. However, a major drawback of this approach is that it only provides estimates of the pre-specified quantiles instead of characterizing the forecast distribution.

The GP-Copula approach in [265] can be thought of as a generalization of the DeepAR [52] model to a multivariate setting. A Gaussian copula process modelling of the forecasts provides a principled way to deal with the difficulty associated with different scales of individual time-series [52, 53]. A low-rank covariance structure is designed to capture the relationship among the forecasts with reduced computational complexity. The resulting algorithm is shown to scale well for datasets containing a few thousand time-series.

Generalizing the adoption of linear state-space model in the DeepState [54] algorithm, Kurle et al. propose to use a more flexible switching Gaussian linear dynamical system model in [266] for multivariate forecasting. The Rao-Blackwellised particle filter [89] is employed for tractable inference of the hidden states. The conditionally linear-Gaussian emission model might not be adequate to represent the generation of complex, typically non-Gaussian, multivariate time-series data. Hence, an auxiliary variable approach, which allows for more complex nonlinear emission models with a decoder neural network is adopted. An encoder, similar to the Ladder VAE [271], is used for designing an efficient proposal mechanism with

dependencies in the same direction as the generative model. The parameters of the overall architecture are learned by variational lower bound maximization approach.

Use of normalizing flow [267] constitutes another research direction in probabilistic forecasting. Normalizing flow is a powerful, scalable, and general approach, which allows modelling of arbitrarily complex distributions of correlated, high-dimensional data by applying a sequence of deterministic, invertible transformations to a simple initial distribution. In [268], a flexible, nonlinear, and bijective transformation of the linear-Gaussian emission mechanism in a SSM is considered as the generative process of multivariate time-series data. As in normalizing flow models, the resulting likelihood function can be computed in closed form using the change of variables formula. Similar to [54], an RNN with covariates as inputs is used to model the time-varying parameters of the SSM. In addition to impressive probabilistic forecasting, this approach shows improved capability in dealing with missing observations. Another approach in [269] applies more sophisticated transformations, such as Real NVP [272] and *masked autoregressive flow* (MAF) [273], to the RNN states for representing forecasting uncertainty. The resulting algorithms achieve state-of-the-art performance on several benchmark multivariate time-series datasets. Aside from these forecasting approaches, normalizing flow has been combined with RNNs for other tasks such as stochastic video generation [274] and modelling the density of sequential data [275].

In addition to the restrictive assumptions on the univariate forecast distributions in some of these approaches [52–54], extending them to the multivariate setting is often not straightforward due to the increased computational complexity. Since the probabilistic algorithms mainly focus on learning the forecasting distributions, they perform poorly in terms of the point forecasting metrics. Another major drawback of all of these probabilistic forecasting techniques is that when applied in a spatio-temporal setting, these models cannot utilize the graph information for better learning.

2.3.4 Spatio-Temporal Forecasting Models

With the advent of graph neural networks [37, 62], there has been extensive research effort towards developing spatio-temporal forecasting algorithms that are capable of modelling the complex spatial relationships among multivariate time-series to achieve more accurate

predictions. These algorithms either utilize an observed graph adjacency matrix [50, 276] or learn a plausible topology from the data [51, 277, 278]. Graph convolution [37, 62] or attention [38] mechanism is applied to capture complex topological structures for representing spatial dependency in conjunction with various deep learning modules for handling the temporal dynamics. We broadly categorize these algorithms based on their temporal neural network architectures, including recurrent networks [133, 250], temporal convolutions [263, 279], and attention mechanisms [258].

The *Temporal Graph Convolutional Network* [280] (TGCN) combines a GCN [37] with a GRU [250] so that the input to the GRU layers at any time instant is obtained by performing a spatial aggregation of the multivariate time-series at that instant using a GCN layer. A similar architecture using an LSTM [133] is proposed in [281] for classification of skeleton-based action sequences. In the *Traffic Graph Convolutional Recurrent Neural Network* [282] (TGCRNN), Cui et al. incorporate additional spatial constraints, depending on whether traffic at one node can be impacted by another node within a specific time, for designing the adjacency matrix.

Li et al. propose a more general approach, termed Diffusion Convolutional Recurrent Neural Network [50] (DCRNN). This model use an encoder-decoder model for *sequence to sequence* forecasting using multi-layer Diffusion Convolutional GRUs (DCGRU). The DCGRU module is constructed by replacing the feedforward matrix multiplications by a graph diffusion convolution for each cell of a GRU as follows:

$$r_t = \sigma\left(\Theta_r \star_{\mathcal{G}} [y_t, x_{t-1}] + b_r\right) , \tag{2.85}$$

$$u_t = \sigma\left(\Theta_u \star_{\mathcal{G}} [y_t, x_{t-1}] + b_u\right) , \tag{2.86}$$

$$c_t = \tanh\left(\Theta_c \star_{\mathcal{G}} [y_t, (r_t \odot x_{t-1})] + b_c\right) , \tag{2.87}$$

$$x_t = u_t \odot x_{t-1} + (1 - u_t) \odot c_t \, . \tag{2.88}$$

Here, r_t, u_t, and c_t are the outputs of the reset, update, and candidate activation gates of the DCGRU respectively. $[\cdot, \cdot]$ stands for column-wise concatenation operation, $\odot$ represents elementwise multiplication, and Θ-s and b-s are the learnable weights and biases respectively. r_t and u_t use sigmoid activation function, whereas c_t is computed using a hyperbolic tangent nonlinearity. The overall computation in a DCGRU layer can be summarized as $x_t = \text{DCGRU}(y_t, x_{t-1})$, where y_t and x_t denote the input and the output to

the layer at time step t. The K-th order diffusion convolution of input graph signal X on (possibly) directed graph $\mathcal{G}$ with adjacency matrix A is denoted by $\star_{\mathcal{G}}$, and is defined as:

$$\Theta \star_{\mathcal{G}} X = \sum_{k=0}^{K-1} \left(T_k(D_O^{-1}A)X\theta_{k,O} + T_k(D_I^{-1}A^T)X\theta_{k,I} \right) , \tag{2.89}$$

where, $D_O = \text{diag}(A\mathbf{1})$ and $D_I = \text{diag}(A^T\mathbf{1})$ are the out-degree and the in-degree matrices of graph $\mathcal{G}$ respectively. The k-th order Chebyshev polynomial is denoted by $T_k(\cdot)$. The learnable parameters are grouped into $\Theta = \{\theta_{k,O}, \theta_{k,I}\}_{k=0}^{K-1}$. Eq. (2.89) contains polynomial terms of both (normalized) row and column adjacency matrices, which facilitates efficient, bidirectional modelling of traffic patterns on real-world datasets. The overall encoder-decoder architecture is trained to minimize a suitable point forecasting criterion such as *mean absolute error* (MAE) of the predictions using scheduled sampling [283].

Huang et al. propose an extension of DCRNN by accommodating more flexible spatial aggregation through a rank influencing mechanism [284]. The rank influence factor matrix term is similar to graph attention [38] and its inclusion results in improved performance at the expense of a moderate increase of computational complexity.

The *Multiple Residual Recurrent Graph Neural Network* [285] models recent and various periodic temporal dependencies by employing several graph convolutional recurrent modules with additional residual shortcut path, gates and hop connections.

The *Adaptive Graph Convolutional Recurrent Network* [51] (AGCRN) model improves the graph-based recurrent architecture by equipping it with *node adaptive parameter learning* (NAPL) for efficient graph convolution and a data-adaptive graph learning module. The AGCRN architecture is formed by stacking Adaptive Graph Convolutional GRUs (AGCGRU). The AGCGRU has a learnable node embedding matrix $E \in \mathbb{R}^{N \times d_e}$, whose i-th row denotes the embedding for the i-th node in the graph and the embedding dimensionality $d_e \ll N$, the number of nodes. The learnable adjacency matrix is formed as:

$$A_{adap} = \text{softmax}\left(EE^T\right) , \tag{2.90}$$

where, the softmax function is applied row-wise. The NAPL enhanced adaptive graph

convolution of K-th order is defined as:

$$Z_j^n = \sum_{k=0}^{K-1} \left(T_k(A_{adap})X\right)_i^n \left(E_\ell^n \theta_{k_{i,j}}^\ell\right)_j^i + E_\ell^n b_j^\ell \,, \tag{2.91}$$

where, $X \in \mathbb{R}^{N \times d_i}$ is the input and $Z \in \mathbb{R}^{N \times d_o}$ is the output of the NAPL-GCN layer. The subscripts and superscripts indicate the Einstein summation notation. The node specific weights and biases are formed based on a matrix factorization approach using the node embedding matrix E. The learnable parameters are $\{\theta_k \in \mathbb{R}^{d_e \times d_i \times d_o}\}_{k=0}^{K-1}$ and $b \in \mathbb{R}^{d_e \times d_o}$. This NAPL-GCN layer is used to construct each gate in the AGCGRU module.

Although the graph based recurrent models achieve impressive performance on several traffic datasets, RNN approaches are known to have difficulty in modelling long-term temporal dependencies and to have a long training time. As an alternative approach, the *Spatio-Temporal Graph Convolutional Network* [276] (STGCN) model uses graph convolution for spatial aggregation and 1-D causal convolutions with Gated Linear Units (GLU) to process the temporal features. The *Long Short-term Graph Convolutional Network* [286] (LSGCN) approach can be viewed as a generalization of the STGCN model. Compared to the STGCN, which uses convolutions in both space and time, the main novelty in the LSGCN model is to integrate a spatial attention with gated temporal convolutions. The Graph WaveNet [287] (GWN) stacks graph convolutional layers with dilated causal convolutions [263], whose receptive field grows exponentially with the number of layers. In addition, the GWN model has the capability of learning a graph from the time-series data and using it to improve forecasting.

The *Multivariate Time-series forecasting with Graph Neural Network* [288] (MTGNN) approach combines a directed graph learning module, graph convolutional layers, and temporal convolutions using residual and skip connections. Similar ideas are explored in the design of the *Structure Learning Convolutional Neural Network* (SLCNN) [277], which strives to extract global and local temporal patterns by two separate learnable adjacency matrices. The *3D-Temporal Graph Convolutional Networks* [289] (3D-TGCN) approach ignores the graph structure based on geographical data and learns an adjacency matrix based on the similarities among the time-series on different nodes. These similarities are computed using the Dynamic Time Warping [290] (DTW) algorithm. The 3D convolutional architecture allows better fusing of the spatial and temporal relationships. In

the *Dynamic Spatial-Temporal Graph Convolutional Network* [291] (DSTGCN), Diao et al. propose to capture time-varying spatial dependencies by learning a dynamic graph Laplacian matrix, with the goal of achieving better modelling of the fluctuations of the spatio-temporal data. Gated CNNs are used for extracting the temporal features. The *Spatial-Temporal Synchronous Graph Convolutional Network* [292] (STSGCN) model constructs a localized spatio-temporal graph by connecting each node with itself at the previous and the next time steps. Subsequently, this graph is used for several convolutions to obtain spatial and temporal embeddings, which are aggregated to form the forecasts. Most of these temporal convolution based architectures exhibit favourable computational requirements compared to the recurrent models.

Another avenue for capturing long-term dependency in the time-series data is explored by integrating various types of attention mechanisms with spatial aggregations. The *Attention Spatial-Temporal Graph Convolutional Networks* [293] (ASTGCN) architecture is composed of three independent components, which aim to learn recent temporal patterns, daily seasonality, and weekly seasonality. Each component contains spatial and temporal attention modules, whose output is then aggregated using graph convolutions. In the *Spatial-Temporal Graph to Sequence* [294] (STG2Seq) model, a long-term encoder and a short-term encoder are formed using separate *Gated Graph Convolutional Modules* (GGCMs). The output forecast is obtained by applying both temporal and channel-wise attention. Multiple spatio-temporal attention blocks are stacked to form an encoder-decoder model, referred to as the *Graph Multi-Attention Network* [295] (GMAN). A transform attention layer, which conveys the encoder's learned features to the decoder, is utilized to circumvent the effect of error propagation, typically observed in recurrent encoder-decoder architectures. This results in improved accuracy for long horizon forecasts. The GMAN model relies on existing node embedding algorithms, such as node2vec [296], to obtain structural representations, which are incorporated in the attention modules. The *Attention-based Periodic Temporal neural Network* [297] (APTN) consists of three components; a recurrent module with temporal skip connections for learning the periodic patterns, an encoder with spatial attention, and a decoder with temporal attention. The *Spatio-Temporal GRaph ATtention* [298] (STGRAT) is a sophisticated, recent encoder-decoder model, which shows high accuracy for larger traffic datasets. Pre-trained node embeddings from the Line [299] algorithm are used as input of the spatial attention

module, whose output is fed to the temporal attention blocks in the encoder and decoder. Another model, termed the *Spatial-Temporal Transformer Network* [300] (STTN), adapts the Transformer [258] architecture for improving the long-range spatio-temporal forecasting. Although these attention-based approaches often outperform the convolutional architectures, they typically have considerably higher number of learnable parameters compared to the convolution-based models.

There are some deep learning models that do not fall into any of these three categories discussed above. Among them, Oreshkin et al. introduce the *Fully Connected GAted Graph Architecture* [278] (FC-GAGA) as a generalization of the N-BEATS [49] model for spatio-temporal forecasting. The FC-GAGA model is endowed with a graph learning module in each layer and for each node, this learned graph is used to weigh the historical data of all other nodes. Subsequently a gating mechanism is applied; the gated outputs for all nodes are stacked together and fed to fully connected residual blocks. Use of such fully connected residual architecture results in a considerably lower training time and memory requirements. The *Spatio-Temporal U-Network* [301] (ST-UNet) model adapts the U-Net [302] architecture, which was originally proposed for image segmentation. The use of the U-Net model along with spatial pooling and temporal downsampling allows the ST-UNet model to capture the spatio-temporal relationships at multiple scales, which might be beneficial for larger traffic datasets.

All of the spatio-temporal models, discussed in this section, mainly focus on designing novel deep learning architectures for the spatio-temporal data and are trained end to end to optimize a point forecasting objective. As a result, despite their impressive performance in the point forecasting task, none of these algorithms is capable of characterizing the uncertainty of the provided predictions.

2.3.5 Stochastic RNNs and Parameter Inference in State-Space Models

Several avenues for incorporating stochasticity in recurrent architectures for improved unsupervised learning of high-dimensional sequences have been explored in the literature. In [223], Boulanger et. al. strive to capture the harmonic and rhythmic probabilistic patterns of polyphonic music scores using a RNN-RBM model. This architecture can be

thought of a sequence of conditional *restricted Boltzmann machines* (RBMs) [303], whose parameters are evolving in time and are represented by the output sequence of a deterministic RNN.

The stochastic RNN in [224] considers a latent variable approach for sequence modelling in a variational inference setting. The encoder and the decoder are parameterized using modified RNNs with an additional latent variable input. Compared to the mixture density approach in [251], incorporation of such latent variables enhances the representation capability of the architecture. A similar idea is pursued in the design of the variational RNN [304], which has a *variational autoencoder* (VAE) [305] with shared parameters for each time-step of the sequence data. Compared to standard RNNs, this architecture enhances flexibility in learning complex dependencies present in highly structured data, since its probabilistic sequence generative mechanism has an additional source of randomness in the form of the latent variable. In contrast to [224], this model benefits by allowing the prior distribution of the latent variable sequence to have some temporal structure.

A considerably different and more complex approach in [306] proposes sequence modelling by stacking a SSM on top of a RNN such that the deterministic hidden states of the RNN acts as inputs to the stochastic state-transition in the SSM. Skip connections from the RNN states to the observations introduce additional coupling between the RNN and the SSM. Following the probabilistic dependency structure, the inference distribution of the hidden states is assumed to have a backward-recursive factorization and is parameterized using a backward RNN. The main advantage of this approach for modelling longer sequences arises from the clear separation of deterministic and stochastic units in the architecture, which allows improved posterior inference via smoothing.

For temporal modelling of video data, the *Kalman variational autoencoders* (KVAE) [307] model aims to separate the low-dimensional dynamics from a complex object recognition module. A linear-Gaussian SSM with time-varying parameters represents the temporal dynamics, whereas its emissions are used as input in the decoder recognition network. An efficient variational inference procedure, which exploits the analytical tractability of the filtering and smoothing distributions of the hidden state in a linear-Gaussian setting, is derived for parameter inference. The resulting algorithm shows impressive performance in both synthetic video generation and missing data imputation tasks. Allowing for a

time-varying, potentially nonlinear dynamics in the latent space, the *deep variational Bayes filter* (DVBF) [308] strives to achieve reliable system identification. Since this approach focuses more on having richer latent representations instead of data compression, it can provide plausible long-term predictions in some settings.

The *recurrent Gaussian process* (RGP) [309] model adopts GP-priors for auto-regressive state transition and nonlinear emission. A sequential recognition model based on RNN is proposed to reduce the parameter complexity of the inference distribution. Another approach combining a SSM and with GP-prior is proposed in [310]. In comparison to [309], this work builds a more flexible variational distribution, which allows better modelling of temporal correlations between latent states.

Several earlier approaches propose use of SMC algorithms as a key ingredient for maximum likelihood estimation [311, 312] or Bayesian inference [313] of SSM parameters. More recent advancements [86, 225–227] employ various neural network models to parameterize the SSM and/or the proposal distribution. In [86], an adaptive proposal mechanism inside the SMC framework is implemented using a combination of recurrent architectures and a mixture density network [314]. Parameter learning resorts to a variational inference approach, which utilizes the *reparameterization trick* [305] to compute the stochastic gradient of the *evidence lower bound* (ELBO) for minimizing the discrepancy between the true posterior and the proposal distributions of the unobserved state.

The *Auto Encoding SMC* (AESMC) [225] algorithm considers joint learning of the model and the proposal distribution via maximization of the lower bound to the log marginal likelihood of observed sequential data. Variational RNN [304] is used to parameterize the generative model and the proposal distribution. The ELBO formulation adopted in this work stems from the *Importance Weighted Auto Encoders* (IWAE) [315] and an efficient sequential implementation of the importance sampling is achieved by the incorporation of the SMC scheme. This idea of optimizing the log likelihood estimate of the observed data using a particle filter is explored independently and concurrently in [226, 227]. In addition, [226] provides a characterization of the asymptotic properties of the resulting Monte Carlo objective. Both [225] and [226] derive the Monte Carlo objective using Jensen's inequality on the expectation of the log marginal likelihood estimate computed using an SMC algorithm, whereas [227] obtains the same objective as a lower bound to the

exact ELBO for the variational SMC family. Various other algorithms combining RNN and SMC are proposed for language modelling [316], visual localization [317], and sequence prediction [318].

Since our work on time-series forecasting in Chapter 5 considers a probabilistic setting using an RNN, it is related to these approaches to some extent. However, there are several important differences. Most of the approaches discussed in this section focus on learning SSM based generative mechanisms for sequence data and are evaluated using the ELBO for the unobserved test data for this generation task. In contrast, the Monte Carlo objective function derived in our work directly addresses the probabilistic forecasting task. For the Bayesian inference of the hidden states, our forecasting algorithm relies on the particle flow [21] procedure because of its impressive performance in approximating complex, high-dimensional posterior distributions. Since particle flow methods can obtain samples from an approximate posterior, there is no need for variational inference. We can also avoid importance sampling, which involves using a proposal distribution and resampling the particles. This eliminates the need to design suitable proposals using neural architectures and running an SMC algorithm for computing and optimizing the resulting Monte Carlo objective.

2.4 Summary

This chapter presents background material and literature reviews for the three main topics addressed in this thesis. A review of sequential inference reveals that existing techniques [30, 31] for tracking a multi-modal state posterior scale poorly to high dimensions. More sophisticated techniques [15, 21, 26], which work well for high-dimensional state estimation, rely on Gaussian or uni-modal assumptions for the posterior. This motivates the development of novel algorithms in Chapter 3 for efficient inference of multi-modal filtering distributions .

In recent years, a plethora of research has been conducted into applying machine learning algorithms on graph structured data. The emergence of advanced GNN models has resulted in improved accuracy for node classification [37,38] and link prediction [39,40]. However, we observe that most of these approaches treat the observed graph as ground truth, whereas the observed topology might be susceptible to estimation errors due to noisy observation or

imperfect modelling assumptions. This motivates the derivation of a Bayesian framework for handing the graph topology. Although there are a few recent methods [41, 43] that aim to account for the potential imperfections in the observed topology, these approaches have limitations. They cannot address tasks other than node classification and they rely on variational approximations for parametric modelling of the underlying graph structure. This motivates the development of Bayesian GCN algorithms in Chapter 4 using a non-parametric graph learning technique, which alleviates these difficulties.

A detailed survey of time-series forecasting techniques shows that, although the recent deep learning models [48–51] outperform statistical techniques in terms of point forecasting accuracy for large scale real-world datasets, these models cannot obtain confidence intervals for the forecasts. Existing probabilistic forecasting approaches [52, 54, 265] disregard the spatial relationship among the time-series, when applied to a spatio-temporal setting. This motivates the design of probabilistic spatio-temporal forecasting algorithm in Chapter 5 of this thesis.

Chapter 3
Sequential Inference in Presence of Gaussian Mixture Noise Models

3.1 Introduction

In this chapter, we consider the task of high-dimensional nonlinear filtering in the presence of Gaussian mixture distributed noise. In this setting, the filtering distribution becomes multi-modal [29, 30]. Many sophisticated particle filters [22, 23, 26], which show impressive performance in high-dimensions, are not particularly suitable for this scenario, since their proposal mechanisms implicitly depend on Gaussian assumptions regarding the predictive and the posterior distributions. Although the particle flow filters [21, 105, 106, 114] have emerged as an effective alternative to particle filters in high-dimensions, numerical tractability demands the Gaussian or log-concave assumption on the posterior distribution. Sophisticated sequential MCMC approaches [15, 61] are no exception, since they often require the posterior to be log-concave for applying manifold Hamiltonian Monte Carlo [129] based refinement steps. Such restrictions lead to deterioration in the performance of these approaches when they are employed to track multi-modal posterior distributions.

Several works [28–30] employ Gaussian mixtures to approximate non-Gaussian posteriors in general nonlinear HMMs. However, they are unfit for inference in highly nonlinear models and/or high-dimensions, since they rely on linearization of dynamic and measurement models or employ poor proposal mechanisms. In [31], an approximate particle flow is derived for the case of a Gaussian mixture model prior and a linear Gaussian measurement model. However, this approach scales poorly to high-dimensions as the computation of the flow requires inversion of a matrix whose size grows rapidly with respect to the state dimensionality.

In this chapter, we develop a) a particle flow algorithm that combines exact particle flow with the extended Kalman filter implementation of the Gaussian sum filter to address the case where the dynamic and measurement models can be approximated by a nonlinearity with additive noise distributed according to a Gaussian mixture; b) a particle flow particle filter

for the same setting, which considers importance sampling in an extended state-space by reformulating the HMM as a switching state space model and designing an efficient proposal distribution based on invertible particle flow [26]; and c) a sequential MCMC algorithm, which uses the flow-based proposal in conjunction with measurement-driven sampling of the auxiliary switching variables in its joint draw step for effective traversal of the multi-modal posterior distribution.

The rest of the chapter is organized as follows. In Section 3.2, we specify the problem we address. We present the proposed particle flow filter, particle flow particle filter, and sequential MCMC algorithm for the Gaussian mixture noise models in Sections 3.3, 3.4, and 3.5 respectively. Simulation examples and results are presented and discussed in Section 3.6. The contributions of this chapter are summarized in Section 3.7.

3.2 Problem Statement

We consider the discrete-time filtering problem where our goal is to track the marginal posterior distribution $p(x_k|z_{1:k})$ recursively with time k, starting from an initial probability density function $p(x_0)$. The unobserved state at time k is denoted by x_k and $z_{1:k} = \{z_1, \cdots, z_k\}$ is a sequence of measurements up to time step k. The dynamic and measurement models are specified as:

$$x_k = g_k(x_{k-1}) + v_k \quad \text{for } k \geqslant 1\,, \tag{3.1}$$

$$z_k = h_k(x_k) + w_k \quad \text{for } k \geqslant 1\,. \tag{3.2}$$

Here $g_k : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x}$ is the state-transition function of the hidden state $x_k \in \mathbb{R}^{d_x}$, and the measurement $z_k \in \mathbb{R}^{d_z}$ is generated conditioned on the current state x_k through a potentially nonlinear measurement model $h_k : \mathbb{R}^{d_x} \to \mathbb{R}^{d_z}$. We assume that $h_k(\cdot)$ is a $\mathcal{C}^1$ function, i.e., $h_k(\cdot)$ is a differentiable function whose first derivative is continuous. The additive process and measurement noises are denoted by $v_k \in \mathbb{R}^{d_x}$ and $w_k \in \mathbb{R}^{d_z}$ respectively. We assume that $p(x_0) = \mathcal{N}(x_0; \mu_0, P_0)$ is Gaussian, whereas $v_k \sim \sum_{m=1}^{M} \psi_{k,m}\mathcal{N}(\tau_{k,m}, Q_{k,m})$ and $w_k \sim \sum_{n=1}^{N} \beta_{k,n}\mathcal{N}(\zeta_{k,n}, R_{k,n})$ are white noise processes independent of each other and are distributed according to Gaussian mixtures.

3.3 Particle Flow for GMM Noises

When the process and measurement noises are distributed as Gaussian mixtures, the predictive and posterior distributions of x_k become multi-modal. In this section, we propose a filter that combines exact particle flow with the Gaussian sum filter [30] (discussed in Section 2.1.4) to recursively compute GMM approximations of these distributions. Particles associated with each Gaussian in the mixture representing the predictive distribution are migrated using particle flow to form one component of the mixture representing the posterior. Extended Kalman filters are run in parallel (one for each component of the mixture), but the means of these EKFs are updated using the component means calculated from the migrated particles. The particle flow filter and the parallel EKFs are thus intertwined, with the covariance matrices being computed by the EKF used for particle flow and the means being computed by particle flow used for EKF updates. The algorithm is summarized in Algorithm 3.1.

The filter is initialized at time k=0 by sampling N_p particles from the initial density $\mathcal{N}(\mu_0, P_0)$. As there is only one Gaussian component in the initial density, the parallel EKF prediction step (line 4 in Algorithm 3.1) results in an M-component predictive distribution at k=1. For all subsequent time steps, the approximate predictive distribution has LM Gaussian components, where L denotes the number of components in the filtering distribution of the previous step $k-1$. Resampling of the Gaussian components (lines 5 and 8 in Algorithm 3.1) using Algorithm 2.1 is performed to restrict the number of components in the predictive distribution to L'. In our experiments, we use L'=M and L=N.

The particles associated with each component of the previous step's posterior are propagated through each component of the GMM dynamic model and are thus distributed according to the corresponding components of the predictive distribution for the current time step. The mixture proportions, means and covariances of the components of the predictive distribution are calculated based on parallel EKF predictions (eqs. (2.22), (2.23), and (2.25)). We only keep the particles corresponding to the L' retained Gaussian components of the predictive distribution.

We loop over each component in the resampled Gaussian mixture predictive distribution, applying particle flow to the particles that correspond to the component (lines 12-24 in

Algorithm 3.1 Particle flow for GMM predictive distribution and likelihood (PF-GMM).

Initialization: Draw $\{x_0^i\}_{i=1}^{N_p}$ from the initial probability density $\mathcal{N}(\mu_0, P_0)$. Estimate $\hat{x}_0 = \frac{1}{N_p}\sum_{i=1}^{N_p} x_0^i$. Set $\lambda_0 = 0$.
for $k = 1$ to K **do**
  if k=1 **then**
    Apply parallel EKF prediction: $\{\mu_{k-1}, P_{k-1}\} \to \{\alpha_{k,m}, \bar{\mu}_{k,m}, \bar{P}_{k,m}\}_{m=1}^{M}$.
    Resample GMM components: $\{\alpha_{k,m}, \bar{\mu}_{k,m}, \bar{P}_{k,m}\}_{m=1}^{M} \to \{\alpha_{k,\ell'}, \bar{\mu}_{k,\ell'}, \bar{P}_{k,\ell'}\}_{\ell'=1}^{L'}$.
  else
    Apply parallel EKF prediction: $\{\gamma_{k-1,\ell}, \mu_{k-1,\ell}, P_{k-1,\ell}\}_{\ell=1}^{L} \to \{\alpha_{k,\ell m}, \bar{\mu}_{k,\ell m}, \bar{P}_{k,\ell m}\}_{\ell,m=1}^{L,M}$.
    Resample GMM components: $\{\alpha_{k,\ell m}, \bar{\mu}_{k,\ell m}, \bar{P}_{k,\ell m}\}_{\ell,m=1}^{L,M} \to \{\alpha_{k,\ell'}, \bar{\mu}_{k,\ell'}, \bar{P}_{k,\ell'}\}_{\ell'=1}^{L'}$.
  end if
  Propagate particles: $\eta_{0,m}^i = g_k(x_{k-1}^i) + v_{k,m}$ (if k=1) or $\eta_{0,\ell m}^i = g_k(x_{k-1,\ell}^i) + v_{k,m}$ (if k>1), where $v_{k,m} \sim \mathcal{N}(\tau_{k,m}, Q_{k,m})$ for $i = 1, \cdots, N_p$, $\ell = 1, \cdots, L$, and $m = 1, \cdots, M$.
  Keep only the particles corresponding to the L' retained Gaussian components of the predictive distribution: $\{\eta_{0,m}^i\}_{,m,i=1}^{M,N_p} \to \{\eta_{0,\ell'}^i\}_{\ell,i'=1}^{L',N_p}$ (if k=1) or $\{\eta_{0,\ell m}^i\}_{\ell,m,i=1}^{L,M,N_p} \to \{\eta_{0,\ell'}^i\}_{\ell,i'=1}^{L',N_p}$ (if k>1). Set $\bar{\mu}_{\ell'} = \frac{1}{N_p}\sum_{i=1}^{N_p} \eta_{0,\ell'}^i$ for $\ell' = 1, \cdots, L'$.
  for $\ell' = 1, \cdots, L'$ **do**
    for $n = 1, \cdots, N$ **do**
      for $i = 1, \cdots, N_p$ **do**
        Set $\eta_{0,\ell' n}^i = \eta_{0,\ell'}^i$.
        for $p = 1, \cdots, N_\lambda$ **do**
          Set $\lambda_p = \lambda_{p-1} + \epsilon_p$.
          Calculate $A_{\ell' n}^i(\lambda_p)$ and $b_{\ell' n}^i(\lambda_p)$ from eq. (2.55) and (2.56) with linearization of h_k performed at $\eta_{\lambda_{p-1},\ell' n}^i$, and with $z = z_k - \zeta_{k,n}$, $\bar{\mu} = \bar{\mu}_{\ell'}$, $\bar{P} = \bar{P}_{k,\ell'}$, and $R = R_{k,n}$.
          Migrate particles: $\eta_{\lambda_p,\ell' n}^i = \eta_{\lambda_{p-1},\ell' n}^i + \epsilon_p(A_{\ell' n}^i(\lambda_p)\eta_{\lambda_{p-1},\ell' n}^i + b_{\ell' n}^i(\lambda_p))$
        end for
        Set $x_{k,\ell' n}^i = \eta_{1,\ell' n}^i$.
      end for
    end for
  end for
  Apply parallel EKF update: $\{\alpha_{k,\ell'}, \bar{\mu}_{k,\ell'}, \bar{P}_{k,\ell'}\}_{\ell'=1}^{L'} \to \{\gamma_{k,\ell' n}, \mu_{k,\ell' n}, P_{k,\ell' n}\}_{\ell',n=1}^{L',N}$.
  Set $\mu_{k,\ell' n} = \frac{1}{N_p}\sum_{i=1}^{N_p} x_{k,\ell' n}^i$.
  Estimate $\hat{x}_k = \sum_{\ell'=1}^{L'}\sum_{n=1}^{N} \gamma_{k,\ell' n}\mu_{k,\ell' n}$.
  Resample GMM components: $\{\gamma_{k,\ell' n}, \mu_{k,\ell' n}, P_{k,\ell' n}\}_{\ell',n=1}^{L',N} \to \{\gamma_{k,\ell}, \mu_{k,\ell}, P_{k,\ell}\}_{\ell=1}^{L}$.
  Keep only the N_p particles corresponding to each of the L retained Gaussian components of the posterior distribution: $\{x_{k,\ell}^i\}_{\ell=1,i=1}^{L,N_p}$.
  (Optional) Redraw particles $\{x_{k,\ell}^i\}_{i=1}^{N_p} \sim \mathcal{N}(\mu_{k,\ell}, P_{k,\ell})$ for $\ell = 1, \cdots, L$.
end for

Algorithm 3.1). A separate flow is applied for each of the N components in the mixture representing the observation model, so at the end of this loop we have $L'N$ sets of particles, each representing a different component of the posterior. The mean of each component is estimated using the sample mean of the particles associated with that component. Parallel EKFs update the covariances and proportions of the mixture components (eqs. (2.32) and (2.33)). At each time step, the posterior is approximated by an L component Gaussian mixture, via resampling of Gaussian components (line 28 in Algorithm 3.1).

The most computationally demanding parts of the algorithm are the inverse operations in calculating $A^i_{\ell' n}(\lambda_p)$ and $b^i_{\ell' n}(\lambda_p)$. Since individual flow parameters are calculated for each of the N_p particles, and there are a total of $L'N$ separate flows at each time step with N_λ pseudo time steps, the total computational complexity of the matrix inverse operations is $\mathcal{O}(\ell' N N_p N_\lambda d_z^3)$, where d_z is the measurement dimension. This shows that the computational requirements of our approach scale in the same way to high-dimensions as the exact particle flow filter [21, 105].

3.4 Particle Flow Particle Filter for GMM Noises

In this section, we develop a novel particle flow particle filter to address the setting where the process and measurement noises are distributed as Gaussian mixtures. We observe that the HMM specified by eqs. (3.1) and (3.2) can be alternatively expressed as a switching state space model. We introduce two unobserved scalar-valued discrete random variables $d_k \in \{1, \ldots, M\}$ and $c_k \in \{1, \ldots, N\}$ such that $p(d_k = m) = \psi_{k,m}$ and $p(c_k = n) = \beta_{k,n}$. The d_k and c_k variables are independent for different k and independent of each other. Let $p(x_k|x_{k-1}, d_k{=}m) = \mathcal{N}(x_k|g_k(x_{k-1}) + \tau_{k,m}, Q_{k,m}), \forall 1 \leqslant m \leqslant M$ and $p(z_k|x_k, c_k{=}n) = \mathcal{N}(z_k|h_k(x_k) + \zeta_{k,n}, R_{k,n}), \forall 1 \leqslant n \leqslant N$. The state transition density is then:

$$\begin{aligned} p(x_k|x_{k-1}) &= \textstyle\sum_{m=1}^{M} \psi_{k,m} \mathcal{N}(x_k|g_k(x_{k-1}) + \tau_{k,m}, Q_{k,m}) \,, \\ &= \textstyle\sum_{m=1}^{M} P(d_k = m) p(x_k|x_{k-1}, d_k = m) \,. \end{aligned} \tag{3.3}$$

Similarly, the likelihood is

$$\begin{aligned} p(z_k|x_k) &= \textstyle\sum_{n=1}^{N} \beta_{k,n} \mathcal{N}(z_k|h_k(x_k) + \zeta_{k,n}, R_{k,n}) \,, \\ &= \textstyle\sum_{n=1}^{N} P(c_k = n) p(z_k|x_k, c_k = n) \,. \end{aligned} \tag{3.4}$$

We augment x_k with the unobserved discrete variables d_k and c_k and consider the target joint density to be $p(x_{0:k}, d_{1:k}, c_{1:k}|z_{1:k})$. We require that the importance distribution q factorizes:

$$q(x_{0:k}, d_{1:k}, c_{1:k}|z_{1:k}) = q(x_{0:k-1}, d_{1:k-1}, c_{1:k-1}|z_{1:k-1}) q(x_k, d_k, c_k|x_{0:k-1}, d_{1:k-1}, c_{1:k-1}, z_{1:k}) \,. \tag{3.5}$$

Samples $\{x^i_{0:k}, d^i_{1:k}, c^i_{1:k}\}_{i=1}^{N_p}$ are obtained by augmenting each existing sample,

$$(x^i_{0:k-1}, d^i_{1:k-1}, c^i_{1:k-1}) \sim q(x_{0:k-1}, d_{1:k-1}, c_{1:k-1}|z_{1:k-1}) \,, \tag{3.6}$$

with the new state at time step k,

$$(x^i_k, d^i_k, c^i_k) \sim q(x_k, d_k, c_k|x^i_{0:k-1}, d^i_{1:k-1}, c^i_{1:k-1}, z_{1:k}) \,. \tag{3.7}$$

The target joint density can be expressed as follows:

$$p(x_{0:k}, d_{1:k}, c_{1:k}|z_{1:k}) \propto p(x_k, d_k, c_k|x_{k-1}, d_{k-1}, c_{k-1}) p(z_k|x_k, d_k, c_k) p(x_{0:k-1}, d_{1:k-1}, c_{1:k-1}|z_{1:k-1}) \,. \tag{3.8}$$

Using eqs. (3.5) and (3.8), we can calculate the unnormalized importance weights as:

$$\begin{aligned} \omega^i_k &= \frac{p(x^i_{0:k}, d^i_{1:k}, c^i_{1:k}|z_{1:k})}{q(x^i_{0:k}, d^i_{1:k}, c^i_{1:k}|z_{1:k})}, \\ &\propto \omega^i_{k-1} \frac{p(x^i_k, d^i_k, c^i_k|x^i_{k-1}, d^i_{k-1}, c^i_{k-1}) p(z_k|x^i_k, d^i_k, c^i_k)}{q(x^i_k, d^i_k, c^i_k|x^i_{0:k-1}, d^i_{1:k-1}, c^i_{1:k-1}, z_{1:k})} \,, \end{aligned} \tag{3.9}$$

We design a proposal such that

$$q(x_k, d_k, c_k|x_{0:k-1}, d_{1:k-1}, c_{1:k-1}, z_{1:k}) = q(x_k, d_k, c_k|x_{k-1}, d_{k-1}, c_{k-1}, z_k) \tag{3.10}$$

Algorithm 3.2 Particle flow particle filter for Gaussian mixture noise models (PFPF-GMM).

Initialization: Draw $\{x_0^i\}_{i=1}^{N_p}$ from the prior $\mathcal{N}(\mu_0, P_0)$. Set $\{\omega_0^i\}_{i=1}^{N_p} = \frac{1}{N_p}$, $\{P_0^i\}_{i=1}^{N_p} = P_0$, and $\hat{x}_0 = \sum_{i=1}^{N_p} \omega_0^i x_0^i$. Set $\lambda_0 = 0$.
for $k = 1$ to K **do**
  for $i = 1, \cdots, N_p$ **do**
    Sample $d_k^i = m \in \{1, \cdots, M\}$ with probability $\{\psi_{k,1}, \cdots, \psi_{k,M}\}$.
    Apply EKF prediction: $\{x_{k-1}^i, P_{k-1}^i\} \rightarrow \{m_{k|k-1,m}^i, P_{k,m}^i\}$, using $\mathcal{N}(\tau_{k,m}, Q_{k,m})$.
    Calculate $\bar{\eta}_0^i = g_k(x_{k-1}^i) + \tau_{k,m}$.
    Propagate particle $\eta_0^i = g_k(x_{k-1}^i) + v_{k,m}$, where $v_{k,m} \sim \mathcal{N}(\tau_{k,m}, Q_{k,m})$
    Sample $c_k^i = n \in \{1, \cdots, N\}$ with probability $\{\beta_{k,1}, \cdots, \beta_{k,N}\}$.
    Set $\theta_{mn}^i = 1$.
    for $p = 1, \cdots, N_\lambda$ **do**
      Set $\lambda_p = \lambda_{p-1} + \epsilon_p$.
      Calculate $A_{mn}^i(\lambda_p)$ and $b_{mn}^i(\lambda_p)$ from eq. (2.55) and (2.56) with linearization of h_k performed at $\bar{\eta}_{\lambda_{p-1}}^i$, and with $z = z_k - \zeta_{k,n}$, $\bar{\mu} = \bar{\eta}_0^i$ and $\bar{P} = P_{k,m}^i$, and $R = R_{k,n}$.
      Migrate auxiliary particle: $\bar{\eta}_{\lambda_p}^i = \bar{\eta}_{\lambda_{p-1}}^i + \epsilon_p(A_{mn}^i(\lambda_p)\bar{\eta}_{\lambda_{p-1}}^i + b_{mn}^i(\lambda_p))$.
      Migrate particle: $\eta_{\lambda_p}^i = \eta_{\lambda_{p-1}}^i + \epsilon_p(A_{mn}^i(\lambda_p)\eta_{\lambda_{p-1}}^i + b_{mn}^i(\lambda_p))$.
      $\theta_{mn}^i = \theta_{mn}^i |det(I + \epsilon_p A_{mn}^i(\lambda_p))|$.
    end for
    Set $x_k^i = \eta_1^i$.
    Calculate importance weights:

$$\tilde{\omega}_k^i \propto \omega_{k-1}^i \frac{p(x_k^i | x_{k-1}^i, d_k^i = m) p(z_k | x_k^i, c_k^i = n)}{p(\eta_0^i | x_{k-1}^i, d_k^i = m)/\theta_{mn}^i}$$

  end for
  for $i = 1, \cdots, N_p$ **do**
    Normalize $\omega_k^i = \tilde{\omega}_k^i / \sum_{s=1}^{N_p} \tilde{\omega}_k^s$.
    Apply EKF update: $\{m_{k|k-1,m}^i, P_{k,m}^i\} \rightarrow \{m_{k|k}^i, P_k^i\}$ using $\mathcal{N}(\zeta_{k,n}, R_{k,n})$.
  end for
  Estimate $\hat{x}_k = \sum_{i=1}^{N_p} \omega_k^i x_k^i$.
  (Optional) resample particles : $\{x_k^i, P_k^i, \omega_k^i\}_{i=1}^{N_p}$ to obtain $\{x_k^i, P_k^i, \frac{1}{N_p}\}_{i=1}^{N_p}$.
end for

is satisfied, so that we only need to store the particles and weights from the previous time step, instead of the full trajectories. For the filtering problem, our focus is the marginal

posterior, and we approximate this as

$$p(x_k|z_{1:k}) \approx \sum_{i=1}^{N_p} \omega_k^i \delta_{x_k^i}(x_k)\,, \tag{3.11}$$

where $\delta_a(\cdot)$ is the Dirac delta-function centred at a. From the dynamic model, we have

$$p(x_k, d_k, c_k|x_{k-1}, d_{k-1}, c_{k-1}) = p(d_k)p(c_k)p(x_k|x_{k-1}, d_k)\,, \tag{3.12}$$

and similarly the measurement model satisfies

$$p(z_k|x_k, d_k, c_k) = p(z_k|x_k, c_k)\,. \tag{3.13}$$

We design q such that

$$q(x_k, d_k, c_k|x_{k-1}, d_{k-1}, c_{k-1}, z_k) = q(d_k)q(c_k)q(x_k|x_{k-1}, d_k, c_k, z_k) \tag{3.14}$$

is satisfied. We choose $q(d_k) = p(d_k)$, $q(c_k) = p(c_k)$, and conditioned on $(d_k = m, c_k = n)$, we construct the proposal density $q(x_k^i|x_{k-1}^i, d_k^i, c_k^i, z_k)$ using eq. (2.59) by performing an invertible particle flow [26] (details in Section 2.1.7) using the m-th Gaussian component of the dynamic model and the n-th Gaussian component of the measurement model. The resulting proposal can be evaluated as

$$q(x_k^i|x_{k-1}^i, d_k^i{=}m, c_k^i{=}n, z_k) = \frac{p(\eta_0^i|x_{k-1}^i, d_k^i = m)}{|\prod_{p=1}^{N_\lambda} \det(I + \epsilon_p A_{mn}^i(\lambda_p))|}\,. \tag{3.15}$$

Using eqs. (3.12), (3.13), (3.14), and (3.15), the importance weight update step in eq. (3.9) can be rewritten as

$$\omega_k^i \propto \omega_{k-1}^i \frac{p(x_k^i|x_{k-1}^i, d_k^i)p(z_k|x_k^i, c_k^i)}{q(x_k^i|x_{k-1}^i, d_k^i, c_k^i, z_k)}\,. \tag{3.16}$$

The resulting PFPF-GMM algorithm is summarized in Algorithm 3.2. The most computationally demanding parts of the algorithm are the matrix inverse operations needed to calculate the flow parameters $A_{mn}^i(\lambda_p)$ and $b_{mn}^i(\lambda_p)$. Since individual flow parameters are calculated for each of the N_p particles at each time step and there are a

total of N_λ discrete pseudo time steps, the total computational complexity of the matrix inverse operations is $\mathcal{O}(N_p N_\lambda d_z^3)$, where d_z is the measurement dimension.

3.5 SmHMC with LEDH for GMM Noises

In order to explore different modes of the posterior efficiently, we consider an extended state-space (x_k, d_k, c_k), as in Section 3.4, and rewrite the joint posterior distribution in eq. (3.8) as

$$\begin{aligned}\pi_k(x_{0:k}, d_{1:k}, c_{1:k}) &= p(x_{0:k}, d_{1:k}, c_{1:k}|z_{1:k})\,, \\ &\propto p(d_k)p(c_k)p(x_k|x_{k-1}, d_k)p(z_k|x_k, c_k)\pi_{k-1}(x_{0:k-1}, d_{1:k-1}, c_{1:k-1})\,, \end{aligned} \tag{3.17}$$

which admits $\pi_k(x_{0:k}) = p(x_{0:k}|z_{1:k})$ as its $x_{0:k}$ marginal. Similar to eq. (2.62), based on the approximate joint posterior $\widehat{\pi}_{k-1}(x_{0:k-1}, d_{1:k-1}, c_{1:k-1})$, of the previous time step $k-1$, we approximate $\pi_k(x_{0:k}, d_{1:k}, c_{1:k})$ as follows:

$$\breve{\pi}(x_{0:k}, d_{1:k}, c_{1:k}) \propto p(d_k)p(c_k)p(x_k|x_{k-1}, d_k)p(z_k|x_k, c_k)\widehat{\pi}_{k-1}(x_{0:k-1}, d_{1:k-1}, c_{1:k-1})\,. \tag{3.18}$$

For this model, in the joint draw step of SMCMC, we adopt the following strategy. First we sample

$$(x^{*(i)}_{k,0:k-1}, d^{*(i)}_{k,1:k-1}, c^{*(i)}_{k,1:k-1}) \sim \widehat{\pi}_{k-1}(x_{0:k-1}, d_{1:k-1}, c_{1:k-1})\,. \tag{3.19}$$

Similar to the spirit of auxiliary particle filtering [87], we design efficient measurement-driven proposal distributions for sampling (d_k, c_k), which allows for efficient traversal of multiple modes of the posterior distribution. This is in contrast to the PFPF-GMM algorithm in Section 3.4, where the switching variables are sampled from their respective priors. To sample $d^{*(i)}_{k,k}$, we use the following proposal:

$$q(d^{*(i)}_{k,k} = m|x^{*(i)}_{k,k-1}, z_k) \propto p(d^{*(i)}_{k,k} = m)p(\bar{\eta}_0^{*(i)}|x^{*(i)}_{k,k-1}, d^{*(i)}_{k,k} = m)p(z_k|\bar{\eta}_0^{*(i)})\,, \tag{3.20}$$

where $\bar{\eta}_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}) + \tau_{k,m}$. Conditioned on $d_{k,k}^{*(i)} = m$, we sample $c_{k,k}^{*(i)}$ from

$$q(c_{k,k}^{*(i)} = n | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)} = m, z_k) \propto p(c_{k,k}^{*(i)} = n) p(z_k | \bar{\eta}_0^{*(i)}, c_{k,k}^{*(i)} = n)\,. \tag{3.21}$$

Then conditioned on $(d_{k,k}^{*(i)} = m, c_{k,k}^{*(i)} = n)$, we calculate $\bar{\eta}_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}) + \tau_{k,m}$ to initialize the auxiliary LEDH particle flow with parameters $(C^{*(i)}, D^{*(i)})$, using the m-th and the n-th Gaussian component of the dynamic and measurement model respectively. Then this flow is applied to the propagated particle $\eta_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}) + v_{k,m}$, where $v_{k,m} \sim \mathcal{N}(\tau_{k,m}, Q_{k,m})$ is the m-th component in the process noise. The proposed particle is generated as: $x_{k,k}^i = \eta_1^{*(i)} = C^{*(i)} \eta_0^{*(i)} + D^{*(i)}$. Using the invertible mapping property, established by the flow, we can calculate

$$q(x_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)} = m, c_{k,k}^{*(i)} = n, z_k) = \frac{p(\bar{\eta}_0^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)} = m)}{|\det(C^{*(i)})|}\,. \tag{3.22}$$

Using eqs. (3.18), (3.20), (3.21) and (3.22), the acceptance rate for the joint draw of $(x_{k,0:k}^{*(i)}, d_{k,1:k}^{*(i)}, c_{k,1:k}^{*(i)})$ using the proposed kernel can be calculated as:

$$\begin{aligned}\rho_1 = \min\Bigg(1, \frac{p(d_{k,k}^{*(i)}) p(c_{k,k}^{*(i)}) p(x_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)})}{q(d_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, z_k) q(c_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)}, z_k)} \; \frac{q(d_{k,k}^{i-1} | x_{k,k-1}^{i-1}, z_k) q(c_{k,k}^{i-1} | x_{k,k-1}^{i-1}, d_{k,k}^{i-1}, z_k)}{p(d_{k,k}^{i-1}) p(c_{k,k}^{i-1}) p(x_{k,k}^{i-1} | x_{k,k-1}^{i-1}, d_{k,k}^{i-1})} \\ \times \frac{|\det(C^{*(i)})| p(\bar{\eta}_0^{i-1} | x_{k,k-1}^{i-1}, d_{k,k}^{i-1}) p(z_k | x_{k,k}^{*(i)}, c_{k,k}^{*(i)})}{p(\bar{\eta}_0^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)}) |\det(C^{i-1})| p(z_k | x_{k,k}^{i-1}, c_{k,k}^{i-1})}\Bigg)\,.\end{aligned} \tag{3.23}$$

For individual refinement of $x_{k,0:k-1}^i$, we use the independent proposal $q_{k,2} = \widehat{\pi}_{k-1}$. We can compute the acceptance rate of the refinement as follows:

$$\begin{aligned}\rho_2 &= \min\left(1, \frac{\breve{\pi}_k(x_{k,0:k-1}^{*(i)}, x_{k,k}^i, d_{k,1:k}^i, c_{k,1:k}^i)}{\widehat{\pi}_{k-1}(x_{k,0:k-1}^{*(i)})} \frac{\widehat{\pi}_{k-1}(x_{k,0:k-1}^i)}{\breve{\pi}_k(x_{k,0:k}^i, d_{k,1:k}^i, c_{k,1:k}^i)}\right) \\ &= \min\left(1, \frac{p(x_{k,k}^i | x_{k,k-1}^{*(i)}, d_{k,k}^i)}{p(x_{k,k}^i | x_{k,k-1}^i, d_{k,k}^i)}\right)\,.\end{aligned} \tag{3.24}$$

Conditioned on $(d_{k,k}^i, c_{k,k}^i)$, the individual refinement of $x_{k,k}^i$ is performed using an mHMC kernel as in [15, 55]. The overall procedure to generate one sample from the resulting SmHMC-GMM (LEDH) algorithm is summarized in Algorithm 3.3. From eq. (3.23), we note that we can discard $\{(d_{k,k}^j, c_{k,k}^j)\}_{j=N_b+1}^{N_b+N_p}$ after every time step k, if we are only

Algorithm 3.3 Composite MH Kernels for state-space models with Gaussian mixture noises, constructed with the manifold Hamiltonian Monte Carlo kernel and the invertible particle flow with LEDH, at the i-th MCMC iteration of k-th time step.
Input: $x_{k,0:k}^{i-1}, \eta_0^{i-1}, d_{k,k}^{i-1}, c_{k,k}^{i-1}, C^{i-1}$.
Output: $x_{k,0:k}^{i}, \eta_0^{i}, d_{k,k}^{i}, c_{k,k}^{i}, C^{i}$.

Joint draw of $x_{k,0:k}^{i}$:

1: Draw $x_{k,0:k-1}^{*(i)} \sim \widehat{\pi}_{k-1}(x_{0:k-1})$.
2: Sample $d_{k,k}^{*(i)} = m \in \{1, \cdots M\}$ from $q(d_k | x_{k,k-1}^{*(i)}, z_k)$.
3: Sample $\eta_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, v_{k,m})$, where $v_{k,m} \sim \mathcal{N}(\tau_{k,m}, Q_{k,m})$.
4: Calculate $\bar{\eta}_0^{*(i)} = g_k(x_{k,k-1}^{*(i)}, \tau_{k,m})$.
5: Sample $c_{k,k}^{*(i)} = n \in \{1, \cdots N\}$ from $q(c_k | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)} = m, z_k)$.
6: Perform invertible particle flow (Algorithm 2.8) $(C^{*(i)}, D^{*(i)})$ by starting the auxiliary particle flow from $\bar{\eta}_0^{*(i)}$ using m-th and n-th component of dynamic and measurement models respectively.
7: Calculate $x_{k,k}^{*(i)} = C^{*(i)} \eta_0^{*(i)} + D^{*(i)}$.
8: Compute the MH acceptance probability $\rho_1 = \min\left(1, \frac{p(d_{k,k}^{*(i)}) p(c_{k,k}^{*(i)}) p(x_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)})}{q(d_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, z_k) q(c_{k,k}^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)}, z_k)} \right.$
$\left. \frac{q(d_{k,k}^{i-1} | x_{k,k-1}^{i-1}, z_k) q(c_{k,k}^{i-1} | x_{k,k-1}^{i-1}, d_{k,k}^{i-1}, z_k)}{p(d_{k,k}^{i-1}) p(c_{k,k}^{i-1}) p(x_{k,k}^{i-1} | x_{k,k-1}^{i-1}, d_{k,k}^{i-1})} \frac{|\det(C^{*(i)})| p(\bar{\eta}_0^{i-1} | x_{k,k-1}^{i-1}, d_{k,k}^{i-1}) p(z_k | x_{k,k}^{*(i)}, c_{k,k}^{*(i)})}{p(\bar{\eta}_0^{*(i)} | x_{k,k-1}^{*(i)}, d_{k,k}^{*(i)}) |\det(C^{i-1})| p(z_k | x_{k,k}^{i-1}, c_{k,k}^{i-1})} \right)$.
9: Accept $x_{k,0:k}^{i} = x_{k,0:k}^{*(i)}$, $\eta_0^{i} = \eta_0^{*(i)}$, $d_{k,k}^{i} = d_{k,k}^{*(i)}$, $c_{k,k}^{i} = c_{k,k}^{*(i)}$, $C^{i} = C^{*(i)}$ and $D^{i} = D^{*(i)}$ with probability ρ_1.
Otherwise set $x_{k,0:k}^{i} = x_{k,0:k}^{i-1}$, $\eta_0^{i} = \eta_0^{i-1}$, $d_{k,k}^{i} = d_{k,k}^{i-1}$, $c_{k,k}^{i} = c_{k,k}^{i-1}$, $C^{i} = C^{i-1}$ and $D^{i} = D^{i-1}$.

10: Individual refinements of $x_{k,0:k}^{i}$ using Algorithm 2.10 given $d_{k,k}^{i}$ and $c_{k,k}^{i}$.
11: Calculate $\eta_0^{i} = (C^{i})^{-1}(x_{k,k}^{i} - D^{i})$.

---

interested in approximating $\pi_k(x_{0:k})$.

3.6 Numerical Experiments and Results

We conduct numerical simulations for two scenarios to evaluate the proposed PF-GMM, PFPF-GMM, and SmHMC-GMM (LEDH) algorithms. The first is a linear scenario, which allows us to compare the performance of the proposed filter with an (almost) optimal solution in the form of the Gaussian mixture model Kalman filter. The second nonlinear

scenario requires a particle filter for accurate state estimates. We compare with an extended Kalman filter derived for Gaussian mixture models (EKF-GMM), the Gaussian Sum Particle Filter (GSPF) [30], an unscented Kalman filter (UKF) [119], the exact Daum Huang (EDH) filter [21] and its localized version (LEDH) [105], the Particle Flow Particle Filters (PFPFs) based on EDH and LEDH [26], and a Bootstrap Particle Filter [13] using 1 million particles. All numerical simulations in this section are executed using an Intel i7-4770K, 3.50GHz CPU and 32GB RAM.

As in [26, 106], we use $N_\lambda = 29$ exponentially spaced step sizes for all particle flow based algorithms so that $\epsilon_p = q^{p-1}\epsilon_1$ is satisfied for $p = 2, \cdots, N_\lambda$. We set the geometric ratio $q = 1.2$, so the initial step size is $\epsilon_1 = \frac{1-q}{1-q^{N_\lambda}} \approx 0.001$. We employ the EKF algorithm to estimate the predictive covariance matrix required to calculate the flow parameters in eq. (2.55) and (2.56). Following [105], the particles are redrawn from the approximate posterior distribution for the EDH and LEDH algorithms after each time step. The UKF uses $2d + 1$ sigma points, where d denotes the dimensionality of the state. For all particle filters employing N_p particles, stratified resampling [72] is performed if the *effective sample size* (ESS) [80] is lower than $\frac{N_p}{2}$. We skip comparison to ensemble Kalman filter (EnKF) [121] and several sophisticated particle filters such as *unscented particle filter* (UPF) [88], *guided sequential Monte Carlo* (GSMC) [22], and *Gaussian particle flow importance sampling* (GFPIS) [23] algorithms, since the PFPF [26] is shown to outperform these approaches. For the EKF-GMM, the PF-GMM, and the GSPF algorithms, resampling of Gaussian components is performed after each prediction and update step and we set $L'{=}M$ and $L{=}N$. In addition, the PF-GMM uses redraw and the GSPF uses resampling of the particles representing each Gaussian component in their GMM approximations of the posterior distribution. The SmHMC based algorithms employ a *manifold Hamiltonian Monte Carlo* kernel [129] for performing the individual refinement for the particles of the current step. Following [15], 20 leapfrog steps with step size 0.5 are used for generating the mHMC proposal. Since the SmHMC [15] is not particularly suitable for multi-modal posteriors that do not satisfy log concavity, we consider the joint inference in the switching state-space model as in the SmHMC-GMM (LEDH) algorithm to establish a suitable SmHMC baseline. The resulting SmHMC-GMM algorithm is an SmHMC variant such that after sampling of (d_k, c_k) using the same proposal distributions as in the SmHMC-GMM (LEDH) in the joint draw, x_k is proposed using the particular

component of the dynamic model specified by d_k. The acceptance rate of the joint draw step in the SmHMC-GMM algorithm is thus computed as:

$$\rho_1 = \min\left(1, \frac{p(d_{k,k}^{*(i)})p(c_{k,k}^{*(i)})q(d_{k,k}^{i-1}|x_{k,k-1}^{i-1},z_k)q(c_{k,k}^{i-1}|x_{k,k-1}^{i-1},d_{k,k}^{i-1},z_k)p(z_k|x_{k,k}^{*(i)},c_{k,k}^{*(i)})}{q(d_{k,k}^{*(i)}|x_{k,k-1}^{*(i)},z_k)q(c_{k,k}^{*(i)}|x_{k,k-1}^{*(i)},d_{k,k}^{*(i)},z_k)p(d_{k,k}^{i-1})p(c_{k,k}^{i-1})p(z_k|x_{k,k}^{i-1},c_{k,k}^{i-1})}\right). \tag{3.25}$$

The individual refinement steps of the SmHMC-GMM algorithm are the same as those of the proposed SmHMC-GMM (LEDH) approach.

3.6.1 Linear Model with GMM Noises

We first consider a linear dynamic and measurement model as follows:

$$x_k = \alpha x_{k-1} + v_k\,, \tag{3.26}$$

$$z_k = x_k + w_k\,, \tag{3.27}$$

where $x_k \in \mathbb{R}^d$ and $z_k \in \mathbb{R}^d$ are the state and observation vectors, respectively. We set $d = 64$ and $\alpha = 0.9$. The process and measurement noises are drawn from GMMs: $v_k \sim \sum_{m=1}^{3} \frac{1}{3}\mathcal{N}(\mu_m \mathbf{1}_{d\times 1}, \sigma_v^2 \mathbf{I}_{d\times d})$, where $\mu_1 = -1$, $\mu_2 = 0$, $\mu_3 = 1$ and $\sigma_v = 1$, $w_k \sim \sum_{n=1}^{3} \frac{1}{3}\mathcal{N}(\delta_n \mathbf{1}_{d\times 1}, \sigma_w^2 \mathbf{I}_{d\times d})$, where $\delta_1 = -5$, $\delta_2 = 0$, $\delta_3 = 5$ and $\sigma_w = 0.1$. The true state starts with $x_0 = \mathbf{0}_{d\times 1}$. All filters are initialized with an initial distribution $p(x_0) = \mathcal{N}(x_0; \mathbf{0}_{d\times 1}, \mathbf{I}_{d\times d})$. The experiment is executed 200 times for 50 time steps.

Table 3.1 summarizes the results, reporting the mean-squared error (MSE) in the state estimation. The EKF-GMM algorithm is optimal in this case except for the error introduced by Gaussian component resampling. We observe that the PF-GMM also attains close to the (almost) optimal performance. The PFPF-GMM method performs only slightly worse than the EKF-GMM algorithm. Although the MSE of the SmHMC-GMM (LEDH) is the same as that of the SmHMC-GMM algorithm, the acceptance rate in the joint draw step (ρ_1) for the SmHMC-GMM (LEDH) is significantly higher than that of the SmHMC-GMM algorithm. This demonstrates the efficacy of the particle flow based joint draw in the SmHMC-GMM (LEDH). The SmHMC based methods with 100 particles achieve better performance than PFPF-GMM algorithm with 200 particles. This shows that MCMC based techniques are in general more suitable than importance sampling in higher dimensions. The GSPF algorithm,

Table 3.1: Average and 5th and 95th sample percentiles of MSE of state estimation, acceptance rates (if applicable) and execution time per step in the linear model with GMM process and measurement noises, based on 100 simulation trials.

Algorithm	No. particles	Avg. MSE	5th and 95th. percentile MSE	Acceptance rate			Exec. time (s)
				ρ_1	ρ_2	ρ_3	
SmHMC-GMM (LEDH)	100	0.010	(0.009, 0.010)	0.225	0.62	0.87	2.25
SmHMC-GMM	100	0.010	(0.009, 0.010)	0.004	0.62	0.89	1.95
PFPF-GMM	200	0.012	(0.011, 0.012)	-	-	-	1.77
PF-GMM	50 per comp.	0.011	(0.010, 0.011)	-	-	-	1.85
GSPF	10^4 per comp.	78.93	(52.86, 105.60)	-	-	-	2.02
EKF-GMM	N/A	0.010	(0.009, 0.010)	-	-	-	0.026
UKF	N/A	1.99	(1.05, 3.16)	-	-	-	0.019
LEDH	500	2.00	(1.05, 3.20)	-	-	-	3.40
EDH	500	1.99	(1.06, 3.16)	-	-	-	0.013
PFPF (LEDH)	500	0.20	(0.02, 0.70)	-	-	-	4.53
PFPF (EDH)	10^5	0.033	(0.012, 0.020)	-	-	-	1.97
BPF	10^6	8.14	(6.08, 11.33)	-	-	-	4.21

which approximates each component of the predictive and posterior densities by a Gaussian distribution by performing importance sampling exhibits poor representation capability in higher dimensions. Our approaches outperform the LEDH and EDH filters and the UKF significantly, since these techniques rely on incorrect Gaussian approximations for the multi-modal predictive distribution and measurement likelihood. The PFPF (LEDH) and PFPF (EDH) algorithms attain lower MSE compared to the corresponding particle flow filters, since the PFPF algorithms employ an importance weight update after the flow. However, the PFPFs calculate predictive covariances using incorrect Gaussian dynamic model assumption and the computation of the particle flow based proposal distribution is reliant on the incorrect Gaussian assumption of the likelihood; therefore they are ill-suited for approximating the multi-modal posterior distribution. The BPF suffers from severe weight degeneracy in this high dimensional state-space, even if 1 million particles are employed.

3.6.2 Nonlinear Model with GMM Noises

We consider a nonlinear dynamical model $g_k : \mathbb{R}^d \to \mathbb{R}^d$ and measurement function $h_k : \mathbb{R}^d \to \mathbb{R}^d$. The c-th dimension of the state-transition function $g_k^c : \mathbb{R} \to \mathbb{R}$ is defined as follows:

$$g_k^c(x_{k-1}) = 0.5x_{k-1}^c + 8\cos(1.2(k-1)) + \begin{cases} 2.5\frac{x_{k-1}^{c+1}}{1+(x_{k-1}^c)^2} & \text{, if } c = 1 \\ 2.5\frac{x_{k-1}^{c+1}}{1+(x_{k-1}^{c-1})^2} & \text{, if } 1 < c < d \\ 2.5\frac{x_{k-1}^{c}}{1+(x_{k-1}^{c-1})^2} & \text{, if } c = d \end{cases} \tag{3.28}$$

The c-th dimension of the measurement function $h_k^c : \mathbb{R} \to \mathbb{R}$ is:

$$h_k^c(x_k) = \frac{(x_k^c)^2}{20}, \ \forall 1 \leqslant c \leqslant d. \tag{3.29}$$

Process noise $v_k \sim \sum_{m=1}^{3} \frac{1}{3}\mathcal{N}(\mu_m \mathbf{1}_{d\times 1}, \sigma_v^2 \mathbf{I}_{d\times d})$, with $\mu_1 = -1$, $\mu_2 = 0$, $\mu_3 = 1$ and $\sigma_v = 0.5$, and measurement noise $w_k \sim \sum_{n=1}^{3} \frac{1}{3}\mathcal{N}(\delta_n \mathbf{1}_{d\times 1}, \sigma_w^2 \mathbf{I}_{d\times d})$, with $\delta_1 = -3$, $\delta_2 = 0$, $\delta_3 = 3$ and $\sigma_w = 0.1$. The true state starts at $x_0 = \mathbf{0}$. For all the filters, we use $p(x_0) = \mathcal{N}(x_0; \mathbf{0}_{d\times 1}, \mathbf{I}_{d\times d})$. The experiment is executed 100 times for 50 time steps. We perform two different experiments with $d = 144$ and $d = 400$.

Table 3.2 shows that while the SmHMC-GMM (LEDH) achieves the same smallest average MSE as the PFPF-GMM algorithm among all evaluated methods in the 144 dimensional scenario, the SmHMC-GMM (LEDH) leads to the smallest average MSE in the 400 dimensional scenario. The comparison of acceptance rates in the joint draw and MSE for these two SmHMC approaches shows that the use of particle flow in the SmHMC-GMM (LEDH) method allows it to explore the state space more efficiently than the SmHMC-GMM algorithm. Although the PF-GMM algorithm performs reasonably well for both $d = 144$ and $d = 400$, the posterior approximation provided by this approach is not statistically consistent because of the nonlinearity in the model, numerical solution of the flow equation, and the approximation errors due to resampling of the Gaussian components. The particle flow algorithms LEDH, EDH, the UKF, and the PFPFs perform poorly as they are better suited for uni-modal posterior distributions. The GSPF approach shows poor performance, since its proposal mechanism is inefficient in a high-dimensional

Table 3.2: Average and 5th and 95th sample percentiles of MSE of state estimation, acceptance rates (if applicable) and execution time per step in the nonlinear model with GMM process and measurement noises, based on 100 simulation trials.

d	Algorithm	No. particles	Avg. MSE	5th and 95th percentile MSE	Acceptance rate			Exec. time (s)
					ρ_1	ρ_2	ρ_3	
144	SmHMC-GMM (LEDH)	100	0.10	(0.06, 0.17)	0.072	0.17	0.74	7.12
	SmHMC-GMM	100	0.12	(0.06, 0.23)	0.005	0.17	0.76	3.3
	PFPF-GMM	200	0.10	(0.07, 0.13)	-	-	-	7.4
	PF-GMM	50 per comp.	0.18	(0.06, 0.79)	-	-	-	12.4
	GSPF	10^4 per comp.	4.53	(3.02, 6.26)	-	-	-	3.7
	EKF-GMM	N/A	2.12	(0.63, 3.80)	-	-	-	0.05
	UKF	N/A	1.30	(0.57, 2.42)	-	-	-	0.15
	LEDH	500	9.05	(1.68, 24.88)	-	-	-	13.9
	EDH	500	11.54	(8.43, 17.67)	-	-	-	0.04
	PFPF (LEDH)	500	5.90	(2.98, 11.92)	-	-	-	19
	PFPF (EDH)	10^5	3.01	(1.06, 8.30)	-	-	-	4.60
	BPF	10^6	0.94	(0.71, 1.43)	-	-	-	9
400	SmHMC-GMM (LEDH)	100	0.09	(0.06, 0.12)	0.018	0.096	0.60	59.5
	SmHMC-GMM	100	0.11	(0.06, 0.21)	0.005	0.085	0.64	13.3
	PFPF-GMM	200	0.11	(0.08, 0.14)	-	-	-	80.2
	PF-GMM	50 per comp.	0.11	(0.06, 0.46)	-	-	-	142.7
	GSPF	10^4 per comp.	5.17	(3.57, 6.87)	-	-	-	9.4
	EKF-GMM	N/A	1.61	(0.07, 3.80)	-	-	-	0.3
	UKF	N/A	5.18	(0.99, 18.72)	-	-	-	1.37
	LEDH	500	23.42	(5.96, 32.32)	-	-	-	152.6
	EDH	500	30.45	(23.70, 46.69)	-	-	-	0.42
	PFPF (LEDH)	500	16.38	(10.73, 27.19)	-	-	-	194
	PFPF (EDH)	10^5	12.27	(7.93, 24.85)	-	-	-	16.8
	BPF	10^6	1.31	(0.95, 1.95)	-	-	-	26.2

state-space. The extended Kalman filter for GMM noises (EKF-GMM) cannot handle the high nonlinearity and leads to large estimation errors. Compared to our approaches, the BPF also has considerably higher MSE, even with 10^6 particles, due to the weight degeneracy in the high-dimensional state space.

3.7 Summary

In this chapter, the problem of estimating the state of a hidden Markov model with Gaussian mixture distributed noises is considered. Existing particle flow, particle flow particle filter, and sequential MCMC algorithms often inherently rely on Gaussian or log-concavity assumptions for the predictive and/or filtering distributions, which makes them ill-suited for the inference of multi-modal state posteriors. Algorithms that rely on the extended Kalman filter or unsophisticated particle filters to approximate the multi-modal filtering distribution using a Gaussian mixture exhibit poor performance in complex, high dimensional settings, as they cannot handle the nonlinearity or show severe weight degeneracy. The proposed PF-GMM, PFPF-GMM, and SmHMC-GMM (LEDH) algorithms demonstrate close to the (almost) optimal performance in a high-dimensional linear scenario, and obtain considerably lower estimation errors compared to existing filtering techniques in the high-dimensional nonlinear example. Although the SmHMC-GMM algorithm attains impressive performance in state-estimation, our experiments show that the invertible particle flow based joint draw step in the proposed SmHMC-GMM (LEDH) method results in a significantly better exploration of the high-dimensional state-space in all cases.

Chapter 4
Bayesian Graph Convolutional Neural Networks

4.1 Introduction

Novel approaches for applying convolutional neural networks to graph-structured data have emerged in recent years. Commencing with the work in [137, 143], there have been numerous developments and improvements. Although these *graph convolutional networks* (GCNs) are promising, the current implementations have limited capability to handle uncertainty in the graph structure, and treat the graph topology as ground-truth information. This in turn leads to an inability to adequately characterize the uncertainty in the predictions made by the neural network.

In contrast to this past work, we employ a Bayesian framework and view the observed graph as a realization from a parametric random graph family. The observed adjacency matrix is then used in conjunction with features and labels to perform joint inference. The results reported in this chapter suggest that this formulation, although computationally more demanding, can lead to an ability to learn more from less data, a better capacity to represent uncertainty, and better robustness and resilience to noise or adversarial attacks.

In this chapter, we present the novel Bayesian GCN framework and discuss how inference can be performed. To provide a concrete example of the approach, we focus on a specific random graph model, the assortative mixed membership block model. We address the task of semi-supervised classification of nodes and examine the resilience of the derived architecture to random perturbations of the graph topology.

In addition, we propose a non-parametric graph inference technique which is incorporated in a Bayesian framework to tackle node and/or edge level learning tasks. Our approach has the following key benefits. First, it generalizes the applicability of the Bayesian techniques outside the realm of parametric modelling. Second, flexible, task specific graph learning can be achieved; this makes effective use of the outputs of existing graph-learning techniques to improve upon them. Third, the graph learning procedure scales well to large graphs, in

contrast to the increased difficulty of parametric approaches. The resulting algorithms offer advantages in node classification and link prediction.

The remainder of the chapter is organized as follows. Section 4.2 reviews some background material on GCNs and a brief introduction to Bayesian Neural Networks is provided in Section 4.3. The proposed Bayesian GCN (BGCN) framework and its instantiation for a node classification task using a specific parametric family of random graphs are presented in Section 4.4. The non-parametric graph learning approach and its incorporation in the BGCN framework for node classification and link prediction are detailed in Section 4.5. Section 4.6 presents and discusses the experimental results on benchmark graph datasets. We summarize the chapter's contribution in Section 4.7.

4.2 Graph Convolutional Networks

Although graph convolutional neural networks have been applied to a variety of inference tasks, in order to make the description more concrete we consider the task of identifying the labels of nodes in a graph. Suppose that we observe a graph $\mathcal{G}_{obs} = (\mathcal{V}, \mathcal{E})$, comprised of a set of N nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. For each node we measure data (or derive features), denoted $\boldsymbol{x}_i \in \mathbb{R}^{d_0 \times 1}$ for node i. These node features are stacked row-wise to obtain the input feature matrix $\mathbf{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]^T$. For some subset of the nodes $\mathcal{L} \subset \mathcal{V}$, we can also measure labels $\mathbf{Y}_{\mathcal{L}} = \{\boldsymbol{y}_i : i \in \mathcal{L}\}$. In a classification context, the label $\boldsymbol{y}_i$ identifies a category; in a regression context $\boldsymbol{y}_i$ can be real-valued. Our task is to use the features $\mathbf{X}$ and the observed graph structure $\mathcal{G}_{obs}$ to estimate the labels of the unlabelled nodes $\mathbf{Y}_{\overline{\mathcal{L}}}$, where $\overline{\mathcal{L}} = \mathcal{V} \setminus \mathcal{L}$ denotes the test set.

A GCN performs this task by performing graph convolution operations within a neural network architecture. Collecting the feature vectors as the rows of a matrix $\mathbf{X}$, the layers of a GCN [37, 62] are of the form:

$$\mathbf{H}^{(1)} = \sigma_0(\mathbf{A}_{\mathcal{G}}\mathbf{X}\mathbf{W}^{(0)}) \tag{4.1}$$

$$\mathbf{H}^{(\ell+1)} = \sigma_\ell(\mathbf{A}_{\mathcal{G}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}), \ \ell \in \{1, 2, ..., L-1\}. \tag{4.2}$$

Here $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ are the learnable weights of the GCN at layer ℓ and we group them to form the set $\mathbf{W} = \{\mathbf{W}^{(\ell)}\}_{\ell=0}^{L-1}$. We denote the output features from layer $\ell-1$ by $\mathbf{H}^{(\ell)} \in \mathbb{R}^{N \times d_\ell}$

and $\sigma_\ell(\cdot)$ is a non-linear activation function at layer ℓ. The matrix $\mathbf{A}_\mathcal{G} \in \mathbb{R}^{N\times N}$ is derived from the observed graph and it determines how the output features are mixed across the graph at each layer. The final output for an L-layer network is $\mathbf{Z}{=}\mathbf{H}^{(L)}$. Training of the weights of the neural network is performed by backpropagation with the goal of minimizing an error metric between the observed labels $\mathbf{Y}_\mathcal{L}$ and the network predictions $\mathbf{Z}_\mathcal{L} = \{\boldsymbol{z}_i : i \in \mathcal{L}\}$ at the nodes in the training set. Performance improvements can be achieved by enhancing the architecture with components that have proved useful for standard CNNs, including attention nodes [38], and skip connections and gates [134, 149].

Although there are many different flavours of GCNs, all current versions process the graph as though it is a ground-truth depiction of the relationship between nodes. This is despite the fact that in many cases the graphs employed in applications are themselves derived from noisy data or modelling assumptions. Spurious edges may be included; other edges may be missing between nodes that have very strong relationships. Incorporating attention mechanisms as in [38] addresses this to some extent; attention nodes can learn that some edges are not representative of a meaningful relationship and reduce the impact that the nodes have on one another. But the attention mechanisms, for computational expediency, are limited to processing existing edges — they cannot create an edge where one should probably exist. This is also a limitation of the ensemble approach of [152], where learning is performed on multiple graphs derived by erasing some edges in the graph.

4.3 Bayesian Neural Networks

There is a rich literature on Bayesian neural networks, commencing with pioneering work [6–8, 319] and extending to more recent contributions [320–323]. Since our work in this chapter is primarily focused on carrying out posterior inference of graph structure to improve graph-based learning tasks and not on developing novel Bayesian approaches for the inference of neural network weights, here we only provide a brief exposition of Bayesian neural networks.

We consider the case where we have training inputs $\mathbf{X} = \{x_1, ..., x_n\}$ and corresponding outputs $\mathbf{Y} = \{y_1, ..., y_n\}$. Our goal is to learn a function $y = f(x)$ via a neural network with fixed configuration (number of layers, activation function, etc., so that the weights are sufficient statistics for f) that provides a likely explanation for the relationship between x

and y. The weights $\mathbf{W}$ are modelled as random variables in a Bayesian approach and we introduce a prior distribution over them. Since $\mathbf{W}$ is not deterministic, the output of the neural network is also a random variable. Prediction for a new input x can be formed by computing an expectation of the random prediction with respect to the posterior distribution of $\mathbf{W}$ as follows:

$$p(y|x, \mathbf{X}, \mathbf{Y}) = \int p(y|x, \mathbf{W}) p(\mathbf{W}|\mathbf{X}, \mathbf{Y}) \, d\mathbf{W} \,. \tag{4.3}$$

The term $p(y|x, \mathbf{W})$ can be viewed as a likelihood; in a classification task it is modelled using a categorical distribution by applying a softmax function to the output of the last layer of the neural network; in a regression task a Gaussian likelihood is often an appropriate choice. The integral in eq. (4.3) is in general intractable. Various techniques for inference of $p(\mathbf{W}|\mathbf{X}, \mathbf{Y})$ have been proposed in the literature, including expectation propagation [320], variational inference [321–323], and Markov Chain Monte Carlo (MCMC) methods [8, 324, 325]. In particular, in [321], it was shown that with suitable variational approximation for the posterior of $\mathbf{W}$, Monte Carlo dropout is equivalent to drawing samples of $\mathbf{W}$ from the approximate posterior and eq. (4.3) can be approximated by a Monte Carlo integral as follows:

$$p(y|x, \mathbf{X}, \mathbf{Y}) \approx \frac{1}{T} \sum_{i=1}^{S} p(y|x, \mathbf{W}_i) \,, \tag{4.4}$$

where S weights $\mathbf{W}_i$ are obtained via dropout.

4.4 BGCN using Parametric Graph Models

We consider a Bayesian approach, viewing the observed graph as a realization from a parametric family of random graphs. We then target inference of the joint posterior of the random graph parameters, weights in the GCN and the node (or graph) labels. Since we are usually not directly interested in inferring the graph parameters, posterior estimates of the labels are obtained by marginalization. The goal is to compute the posterior

probability of labels, which can be written as:

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs}) = \int p(\mathbf{Z}|\mathbf{W}, \mathcal{G}, \mathbf{X}) p(\mathbf{W}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}) p(\mathcal{G}|\lambda) p(\lambda|\mathcal{G}_{obs}) \, d\mathbf{W} \, d\mathcal{G} \, d\lambda \,. \tag{4.5}$$

Here $\mathbf{W}$ is a random variable representing the weights of all layers of a Bayesian GCN over random graph $\mathcal{G}$, and λ denotes the parameters that characterize a family of random graphs. The term $p(\mathbf{Z}|\mathbf{W}, \mathcal{G}, \mathbf{X})$ can be modelled using a categorical distribution by applying a softmax function to the output of the GCN, as discussed previously.

This integral in eq. (4.5) is intractable. We can adopt a number of strategies to approximate it, including variational methods and MCMC. For example, in order to approximate the posterior of weights $p(\mathbf{W}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G})$, we could use variational inference [321–323] or MCMC [8, 324, 325]. Various parametric random graph generation models can be used to model $p(\lambda|\mathcal{G}_{obs})$, for example a stochastic block model [197], a mixed membership stochastic block model [199], or a degree corrected block model [207]. For inference of $p(\lambda|\mathcal{G}_{obs})$, we can use MCMC [326] or variational inference [200].

A Monte Carlo approximation of eq. (4.5) is:

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs}) \approx \frac{1}{V} \sum_{v}^{V} \frac{1}{N_G S} \sum_{i=1}^{N_G} \sum_{s=1}^{S} p(\mathbf{Z}|\mathbf{W}_{s,i,v}, \mathcal{G}_{i,v}, \mathbf{X}) \,. \tag{4.6}$$

In this approximation, V samples λ_v are drawn from $p(\lambda|\mathcal{G}_{obs})$; the precise method for generating these samples from the posterior varies depending on the nature of the graph model. The N_G graphs $\mathcal{G}_{i,v}$ are sampled from $p(\mathcal{G}|\lambda_v)$ using the adopted random graph model. S weight matrices $\mathbf{W}_{s,i,v}$ are sampled from $p(\mathbf{W}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{i,v})$ from the Bayesian GCN corresponding to the graph $\mathcal{G}_{i,v}$.

4.4.1 Assortative Mixed Membership Stochastic Block Model

For the Bayesian GCNs derived in this section, we use an assortative mixed membership stochastic block model (a-MMSBM) for the graph [200, 326] and learn its parameters $\lambda = \{\pi, \beta\}$ using a stochastic optimization approach. The assortative MMSBM, described in the following section, is a good choice to model a graph that has relatively strong community structure (such as the citation networks we study in the experiments section).

It generalizes the stochastic block model by allowing nodes to belong to more than one community and to exhibit assortative behaviour, in the sense that a node can be connected to one neighbor because of a relationship through community A and to another neighbor because of a relationship through community B.

Since $\mathcal{G}_{obs}$ is often noisy and may not fit the adopted parametric block model well, sampling π_v and β_v from $p(\pi, \beta|\mathcal{G}_{obs})$ can lead to high variance. This can lead to the sampled graphs $\mathcal{G}_{i,v}$ being very different from $\mathcal{G}_{obs}$. Instead, we replace the integration over π and β with a maximum a posteriori estimate [327]. We approximately compute

$$\{\hat{\pi}, \hat{\beta}\} = \arg\max_{\beta,\pi} p(\beta, \pi|\mathcal{G}_{obs}) \tag{4.7}$$

by incorporating suitable priors over β and π and use the approximation:

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs}) \approx \frac{1}{N_G S} \sum_{i=1}^{N_G} \sum_{s=1}^{S} p(\mathbf{Z}|\mathbf{W}_{s,i}, \mathcal{G}_i, \mathbf{X}) \,. \tag{4.8}$$

In this approximation, $\mathbf{W}_{s,i}$ are approximately sampled from $p(\mathbf{W}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_i)$ using Monte Carlo dropout over the Bayesian GCN corresponding to $\mathcal{G}_i$. The $\mathcal{G}_i$ are sampled from $p(\mathcal{G}|\hat{\pi}, \hat{\beta})$.

4.4.2 Posterior Inference for a-MMSBM

For the undirected observed graph $\mathcal{G}_{obs} = \{y_{ab} \in \{0, 1\} : 1 \leqslant a < b \leqslant N\}$, $y_{ab} = 0$ or 1 indicates absence or presence of a link between node a and node b. In an MMSBM, each node a has a K dimensional community membership probability distribution $\pi^a = [\pi^{a1}, \cdots, \pi^{aK}]^T$, where K is the number of categories/communities of the nodes and we denote the set of π^a-s for all nodes by $\pi = \{\pi^a\}_{a=1}^N$. For any two nodes a and b, if both of them belong to the same community, then the probability of a link between them is significantly higher than the case where the two nodes belong to different communities [199]. The generative model is described as:

For any two nodes a and b,

- Sample $z_{ab} \sim \pi^a$ and $z_{ba} \sim \pi^b$.
- If $z_{ab} = z_{ba} = k$, sample a link $y_{ab} \sim$ Bernoulli(β^k). Otherwise, $y_{ab} \sim$ Bernoulli(δ).

Here, $0 \leqslant \beta^k \leqslant 1$ is termed the community strength of the k-th community. We group these intra-community link probabilities in a K-dimensional vector $\beta = [\beta^1, \cdots, \beta^K]^T$. The cross-community link probability δ is usually set to a small value. The joint posterior of the parameters π and β is given as:

$$\begin{aligned} p(\pi, \beta|\mathcal{G}_{obs}) &\propto p(\beta)p(\pi)p(\mathcal{G}_{obs}|\pi, \beta)\,, \\ &= \prod_{k=1}^{K} p(\beta^k) \prod_{a=1}^{N} p(\pi^a) \prod_{1 \leqslant a < b \leqslant N} \sum_{z_{ab}, z_{ba}} p(y_{ab}, z_{ab}, z_{ba}|\pi^a, \pi^b, \beta)\,. \end{aligned} \tag{4.9}$$

We use a Beta(η) distribution for the prior of β^k and a Dirichlet distribution, Dir(α), for the prior of π^a, where η and α are hyperparameters.

4.4.3 Expanded Mean Parameterization

Maximizing the posterior of eq. (4.9) is a constrained optimization problem with $\beta^k, \pi^{ak} \in (0, 1)$ and $\sum_{k=1}^{K} \pi^{ak} = 1$. Employing a standard iterative algorithm with a gradient based update rule does not guarantee that the constraints will be satisfied. Hence, we consider an expanded mean parameterization [328] as follows. We introduce the alternative parameters $\theta_{k0}, \theta_{k1} \geqslant 0$ and adopt as the prior for these parameters a product of independent Gamma(η, ρ) distributions. These parameters are related to the original parameter β^k through the relationship $\beta^k = \dfrac{\theta_{k1}}{\theta_{k0} + \theta_{k1}}$. This results in a Beta(η) prior for β^k. Similarly, we introduce a new parameter $\phi_a \in \mathbb{R}_+^K$ and adopt as its prior a product of independent Gamma(α, ρ) distributions. We define $\pi^{ak} = \dfrac{\phi_{ak}}{\sum_{l=1}^{K} \phi_{al}}$, which results in a Dirichlet prior, Dir(α), for π^a. We form two vectors $\theta = [\theta_{10}, \theta_{11}, \cdots, \theta_{K0}, \theta_{K1}]^T$ and $\phi = [\phi_{11}, \cdots, \phi_{1K}, \cdots, \phi_{N1}, \cdots, \phi_{NK}]^T$ to group these alternative parameters. The boundary conditions $\theta_{ki}, \phi_{ak} \geqslant 0$ can be handled by simply taking the absolute value of the update.

4.4.4 Stochastic Optimization and Minibatch Sampling

In this section, we derive the preconditioned gradient ascent algorithm, which is used to maximize the joint posterior in eq. (4.9) over θ and ϕ. If X is a Bernoulli random variable with parameter ρ, we denote its probability mass function $p(X = x|\rho) = \rho^x(1-\rho)^{(1-x)}$ as $\mathcal{B}(x;\rho)$ for $x \in \{0,1\}$, to simplify notation. We first define the following probabilities:

$$\begin{aligned} f_{ab}^{(y)}(k,l) &= p(y_{ab}, z_{ab} = k, z_{ab} = l|\pi^a, \pi^b, \beta)\,, \\ &= \begin{cases} \pi^{ak}\pi^{bk}\mathcal{B}(y_{ab};\beta^k)\,, & \text{if } k = l \\ \pi^{ak}\pi^{bl}\mathcal{B}(y_{ab};\delta)\,, & \text{if } k \neq l, \end{cases} \qquad (4.10) \\ Z_{ab}^{(y)} &= p(y_{ab}|\pi^a, \pi^b, \beta) = \sum_{k=1}^{K}\sum_{l=1}^{K} f_{ab}^{(y)}(k,l)\,, \\ &= \mathcal{B}(y_{ab};\delta^{ab}) + \sum_{k=1}^{K}\Big(\mathcal{B}(y_{ab};\beta^k) - \mathcal{B}(y_{ab};\delta^{ab})\Big)\pi^{ak}\pi^{bk}\,, \qquad (4.11) \end{aligned}$$

and

$$\begin{aligned} f_{ab}^{(y)}(k) &= p(y_{ab}, z_{ab} = k|\pi^a, \pi^b, \beta) = \sum_{l=1}^{K} f_{ab}^{(y)}(k,l)\,, \\ &= \pi^{ak}\Big(\mathcal{B}(y_{ab};\beta^k)\pi^{bk} + \mathcal{B}(y_{ab};\delta^{ab})(1-\pi^{bk})\Big)\,. \qquad (4.12) \end{aligned}$$

Based on these probabilities, we can compute the partial derivatives of the log likelihood of y_{ab} as follows:

$$\begin{aligned} g_{ab}(\theta_{ki}) &= \nabla_{\theta_{ki}} \log p(y_{ab}|\pi,\beta)\,, \\ &= \frac{f_{ab}^{(y)}(k,k)}{Z_{ab}^{(y)}}\left(\frac{|1-i-y_{ab}|}{\theta_{ki}} - \frac{1}{\theta_{k0}+\theta_{k1}}\right), \text{ for } i \in \{0,1\}, \qquad (4.13) \end{aligned}$$

and

$$\begin{aligned} g_{ab}(\phi_{ak}) &= \nabla_{\phi_{ak}} \log p(y_{ab}|\pi,\beta)\,, \\ &= \frac{f_{ab}^{(y)}(k)}{Z_{ab}^{(y)}\phi_{ak}} - \frac{1}{\sum_{l=1}^{K}\phi_{al}}\,. \qquad (4.14) \end{aligned}$$

In many graphs that are appropriately modelled by a stochastic block model, most of the nodes belong strongly to only one of the K communities, so the MAP estimate for many π^a lies near one of the corners of the probability simplex. This suggests that the scaling of different dimensions of ϕ_a can be very different. Similarly, as $\mathcal{G}_{obs}$ is typically sparse, the community strengths β^k are very low, indicating that the scales of θ_{k0} and θ_{k1} are very different. We use preconditioning matrices $G(\theta) = \text{diag}(\theta)^{-1}$ and $G(\phi) = \text{diag}(\phi)^{-1}$ as in [328], to obtain the following update rules:

$$\theta_{ki}^{(t+1)} = \left|\theta_{ki}^{(t)} + \epsilon_t\left(\eta - 1 - \rho\theta_{ki}^{(t)} + \theta_{ki}^{(t)} \sum_{a=1}^{N} \sum_{b=a+1}^{N} g_{ab}(\theta_{ki}^{(t)})\right)\right|, \tag{4.15}$$

$$\phi_{ak}^{(t+1)} = \left|\phi_{ak}^{(t)} + \epsilon_t\left(\alpha - 1 - \rho\phi_{ak}^{(t)} \sum_{b=1, b\neq a}^{N} g_{ab}(\phi_{ak}^{(t)})\right)\right|, \tag{4.16}$$

where $\epsilon_t = \epsilon_0(t+\tau)^{-\kappa}$ is a decreasing step-size.

Direct implementation of eq. (4.15) and (4.16) is $\mathcal{O}(N^2K)$ per iteration, where N is the number of nodes in the graph and K denotes the number of communities. This can be prohibitively expensive for large graphs. We instead employ a stochastic gradient based strategy as follows. For update of θ_{ki}'s in eq. (4.15), we split the $\mathcal{O}(N^2)$ sum over all edges and non-edges, $\sum_{a=1}^{N}\sum_{b=a+1}^{N}$, into two separate terms. One of these is a sum over all observed edges and the other is a sum over all non-edges. We calculate the term corresponding to observed edges exactly (in the sparse graphs of interest, the number of edges is closer to $\mathcal{O}(N)$ than $\mathcal{O}(N^2)$). For the other term we consider a minibatch of 1 percent of randomly sampled non-edges and scale the sum by a factor of 100.

At any single iteration, we update the ϕ_{ak} values for only n randomly sampled nodes ($n < N$), keeping the rest of them fixed. For the update of ϕ_{ak} values of any of the randomly selected n nodes, we split the sum in eq. (4.16) into two terms. One involves all of the neighbors (the set of neighbors of node a is denoted by $\mathcal{N}(a)$) and the other involves all the non-neighbors of node a. We calculate the first term exactly. For the second term, we use $n - |\mathcal{N}(a)|$ randomly sampled non-neighbors and scale the sum by a factor of $\dfrac{N-1-|\mathcal{N}(a)|}{n-|\mathcal{N}(a)|}$ to maintain unbiasedness of the stochastic gradient. Overall the update of the ϕ values involve $\mathcal{O}(n^2K)$ operations instead of $\mathcal{O}(N^2K)$ complexity for a full batch update.

Since the posterior in the MMSBM is very high-dimensional, random initialization often

does not work well. We train a GCN [37] on $\mathcal{G}_{obs}$ and use its softmax output to initialize π and then initialize β based on the block structure imposed by π. The resulting BGCN (MMSBM) algorithm is summarized in Algorithm 4.1.

Algorithm 4.1 BGCN (MMSBM)

Input: $\mathcal{G}_{obs}$, $\mathbf{X}$, $\mathbf{Y}_{\mathcal{L}}$
Output: $p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs})$

1: Initialization: train a GCN to initialize the inference in MMSBM and the weights in the Bayesian GCN.
2: **for** $i = 1 : N_G$ **do**
3: Perform N_b iterations of MMSBM inference to obtain $(\hat{\pi}, \hat{\beta})$.
4: Sample graph $\mathcal{G}_i \sim p(\mathcal{G}|\hat{\pi}, \hat{\beta})$.
5: **for** $s = 1 : S$ **do**
6: Sample weights $\mathbf{W}_{s,i}$ via MC dropout by training a GCN over the graph $\mathcal{G}_i$.
7: **end for**
8: **end for**
9: Approximate $p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs})$ using eq. (4.8).

4.5 BGCN using Non-Parametric Graph Learning

Although the parametric graph modelling approach in Section 4.4 is effective, it has several disadvantages. Choosing an appropriate random graph model is very important and the correct choice can vary greatly for different problems and datasets. For example, the a-MMSBM based graph modelling and inference cannot be performed if the observed graph $\mathcal{G}_{obs}$ contains weighted edges. Bayesian inference of the model parameters is often challenging for large graphs (e.g., batch inference of a-MMSBM parameters scales as $\mathcal{O}(N^2)$, where N is the number of nodes in $\mathcal{G}_{obs}$). Another significant drawback of the technique is that the posterior inference of the 'true' graph $\mathcal{G}$ is carried out solely conditioned on $\mathcal{G}_{obs}$ by marginalizing with respect to the random graph parameters. As a result, any information provided by the node features $\mathbf{X}$ and the training labels $\mathbf{Y}_{\mathcal{L}}$ is completely disregarded for the posterior inference of $\mathcal{G}$. This can be highly undesirable in scenarios where the features and labels are highly correlated with the true graph connectivity.

In order to alleviate these shortcomings, we propose a non-parametric generative model for the adjacency matrix $\mathbf{A}_{\mathcal{G}} \in {\mathbb{R}_+}^{N \times N}$ of the random undirected graph $\mathcal{G}$. $\mathbf{A}_{\mathcal{G}}$ is assumed to

be a symmetric matrix with non-negative entries. We emphasize that our model retains the identities of the nodes and disallows permutations of nodes (permutations of adjacency matrices are not equivalent graphs when node identities are preserved). This characteristic is essential for its use in node and edge level inference tasks. We define the prior distribution for $\mathcal{G}$ as

$$p(\mathcal{G}) \propto \begin{cases} e^{\left(\alpha \mathbf{1}^\top \log(\mathbf{A}_\mathcal{G}\mathbf{1}) - \beta \|\mathbf{A}_\mathcal{G}\|_F^2\right)}, & \text{if } \mathbf{A}_\mathcal{G} \in {\mathbb{R}_+}^{N\times N} \quad \mathbf{A}_\mathcal{G} = \mathbf{A}_\mathcal{G}^\top \\ 0\,, & \text{otherwise}\,. \end{cases} \tag{4.17}$$

The first term in the log prior is a logarithmic barrier on the degree of the nodes which prevents any isolated node in $\mathcal{G}$. The second term is a regularizer based on the Frobenius norm which encourages low weights for the links. α and β are hyperparameters which control the scale and sparsity of $A_\mathcal{G}$.

In a typical graph-based leaning task, we usually have access to some training data $\mathcal{D}$, in addition to $\mathcal{G}_{obs}$. The data $\mathcal{D}$ can include feature vectors, labels, and other available information, depending on the task at hand. In the proposed non-parametric model, the joint likelihood of $\mathcal{G}_{obs}$ and $\mathcal{D}$ conditioned on $\mathcal{G}$ is defined as:

$$p(\mathcal{G}_{obs}, \mathcal{D}|\mathcal{G}) \propto \exp\left(-\|\mathbf{A}_\mathcal{G} \odot \mathbf{D}(\mathcal{G}_{obs}, \mathcal{D})\|_{1,1}\right), \tag{4.18}$$

where $\mathbf{D}(\mathcal{G}_{obs}, \mathcal{D}) \in {\mathbb{R}_+}^{N\times N}$ is a symmetric pairwise distance matrix which encodes the dissimilarity between the nodes. The symbol $\odot$ denotes the Hadamard product and $\|\cdot\|_{1,1}$ denotes the elementwise ℓ_1 norm. The likelihood encourages higher edge weights for the node pairs with lower pairwise distances and vice versa.

Bayesian inference of the graph $\mathcal{G}$ involves sampling from its posterior distribution. The space is high dimensional ($\mathcal{O}(N^2)$, where N is the number of the nodes). Designing a suitable sampling scheme (e.g., MCMC) in such a high dimensional space is extremely challenging and computationally demanding for large graphs. Instead we pursue maximum a posteriori estimation, which is equivalent to approximating the posterior by a point mass at the mode [327]. We solve the following optimization problem:

$$\widehat{\mathcal{G}} = \arg\max_{\mathcal{G}} p(\mathcal{G}|\mathcal{G}_{obs}, \mathcal{D})\,, \tag{4.19}$$

which is equivalent to learning an $N \times N$ symmetric adjacency matrix of $\widehat{\mathcal{G}}$.

$$\mathbf{A}_{\widehat{\mathcal{G}}} = \underset{\substack{\mathbf{A}_{\mathcal{G}} \in {\mathbb{R}_+}^{N \times N}, \\ \mathbf{A}_{\mathcal{G}} = \mathbf{A}_{\mathcal{G}}^\top}}{\arg\min} \|\mathbf{A}_{\mathcal{G}} \odot \mathbf{D}\|_{1,1} - \alpha \mathbf{1}^\top \log(\mathbf{A}_{\mathcal{G}} \mathbf{1}) + \beta \|\mathbf{A}_{\mathcal{G}}\|_F^2 \,. \tag{4.20}$$

The optimization problem in (4.20) has been studied in the context of graph learning from smooth signals. In [172], a primal-dual optimization technique is adopted to solve this problem. However the complexity of this approach scales as $\mathcal{O}(N^2)$, which can be prohibitive for large graphs. Instead, we employ the scalable, approximate algorithm in [180], which has several advantages as follows. First, it can use existing approximate nearest neighbor techniques, as in [175], to reduce the dimensionality of the optimization problem. Second, the graph learning has a computational complexity of $\mathcal{O}(N \log N)$ (the same as approximate nearest neighbor algorithms), while the quality of the learned graph is comparable to the state-of-the-art. Third, if we are not concerned about the scale of the learned graph (which is typical in many learning tasks we consider, since a normalized version of the adjacency or Laplacian matrix is used), the approximate algorithm allows us to effectively use only one hyperparameter instead of α and β to control the sparsity of the solution and provides a useful heuristic for automatically selecting a suitable value based on the desired edge density of the solution.

We use this approximate algorithm for inference of the graph $\mathcal{G}$, which is subsequently used in various learning tasks. Since, we have freedom in choosing a functional form for $\mathbf{D}(\cdot, \cdot)$, we can design suitable distance metrics in a task specific manner. This flexibility allows us to incorporate the graph learning step in diverse problem settings. In the next subsections, we present how the graph learning step can be applied to develop Bayesian algorithms for node classification and link prediction.

4.5.1 Semi-Supervised Node Classification

We revisit the semi-supervised node classification task addressed in Section 4.4. In this setting, we have access to the node attributes $\mathbf{X}$ and the training labels $\mathbf{Y}_{\mathcal{L}}$ in addition to $\mathcal{G}_{obs}$. So, $\mathcal{D} = (\mathbf{X}, \mathbf{Y}_{\mathcal{L}})$. We aim to predict the labels of the nodes in $\overline{\mathcal{L}} = \mathcal{V} \setminus \mathcal{L}$. We propose to incorporate a non-parametric model for inference of $\mathcal{G}$ in the BGCN framework. We aim to compute the marginal posterior probability of the node labels, which is obtained

via marginalization with respect to the graph $\mathcal{G}$ and GCN weights $\mathbf{W}$:

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs}) = \int p(\mathbf{Z}|\mathbf{W}, \mathcal{G}_{obs}, \mathbf{X}) p(\mathbf{W}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}) p(\mathcal{G}|\mathcal{G}_{obs}, \mathbf{X}, \mathbf{Y}_{\mathcal{L}}) \, d\mathbf{W} \, d\mathcal{G} \,. \tag{4.21}$$

As in Section 4.4, the categorical distribution of the node labels $p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs})$ is modelled by applying a softmax function to the output of the last layer of the GCN. The integral in (4.21) cannot be computed in a closed form, so we employ Monte Carlo to approximate it as follows:

$$p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs}) \approx \frac{1}{S} \sum_{s=1}^{S} p(\mathbf{Z}|\mathbf{W}_s, \mathcal{G}_{obs}, \mathbf{X}) \,. \tag{4.22}$$

Here, we learn the *maximum a posteriori* (MAP) estimate $\widehat{\mathcal{G}} = \arg\max_{\mathcal{G}} \; p(\mathcal{G}|\mathcal{G}_{obs}, \mathbf{X}, \mathbf{Y}_{\mathcal{L}})$ and subsequently sample S weight matrices $\mathbf{W}_s$ from $p(\mathbf{W}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \widehat{\mathcal{G}})$ by training a Bayesian GCN using the graph $\widehat{\mathcal{G}}$.

In order to perform the graph learning step, we need to define a pairwise distance matrix $\mathbf{D}$. For this application, we propose to combine the output of a node embedding algorithm and a base classifier to form $\mathbf{D}$:

$$\mathbf{D}(\mathbf{X}, \mathbf{Y}_{\mathcal{L}}, \mathcal{G}_{obs}) = \mathbf{D}_1(\mathbf{X}, \mathcal{G}_{obs}) + \delta \mathbf{D}_2(\mathbf{X}, \mathbf{Y}_{\mathcal{L}}, \mathcal{G}_{obs}) \,. \tag{4.23}$$

Here δ is a hyperparameter which controls the importance of $\mathbf{D}_2$ relative to $\mathbf{D}_1$. The (i, j)'th entries of $\mathbf{D}_1$ and $\mathbf{D}_2$ are defined as follows:

$$D_{1,ij}(\mathbf{X}, \mathcal{G}_{obs}) = \|\boldsymbol{z}_i - \boldsymbol{z}_j\|^2 \,, \tag{4.24}$$

$$D_{2,ij}(\mathbf{X}, \mathbf{Y}_{\mathcal{L}}, \mathcal{G}_{obs}) = \frac{1}{|\mathcal{N}_i||\mathcal{N}_j|} \sum_{k \in \mathcal{N}_i} \sum_{l \in \mathcal{N}_j} \mathbf{1}_{(\hat{c}_k \neq \hat{c}_l)} \,. \tag{4.25}$$

Here, $\boldsymbol{z}_i$ is any suitable embedding of node i and $\hat{c}_i$ is the predicted label at node i obtained from the base classification algorithm. $\mathbf{D}_1$ measures pairwise dissimilarity in terms of the observed topology and features and $\mathbf{D}_2$ summarizes the discrepancy of the node labels in the neighborhood. For the experiments, we choose the Variational Graph Auto Encoder (VGAE) algorithm [39] as the node embedding method to obtain the $\boldsymbol{z}_i$ vectors and use the GCN [37] as the base classifier to obtain the $\hat{c}_i$ values. The neighborhood of the i-th node is

defined as:

$$\mathcal{N}_i = \{j|(i,j) \in \mathcal{E}_{\mathcal{G}_{obs}}\} \cup \{i\}\,.$$

Here, $\mathcal{E}_{\mathcal{G}_{obs}}$ is the set of edges in $\mathcal{G}_{obs}$. With the regard to the choice of the hyperparameter δ, we observe that

$$\delta = \frac{\max\limits_{i,j}\ D_{1,ij}}{\max\limits_{i,j}\ D_{2,ij}}$$

works well in our experiments, although it can be tuned if a validation set is available. Following the approach adopted in the BGCN (MMSBM) algorithm in Section 4.4, we use Monte Carlo dropout [321] to perform variational inference of GCN weights $\mathbf{W}$. The resulting BGCN (Non-Parametric) algorithm is provided in Algorithm 4.2.

Algorithm 4.2 BGCN (NP)

1: **Input:** $\mathcal{G}_{obs}$, $\mathbf{X}$, $\mathbf{Y}_{\mathcal{L}}$
2: **Output:** $p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs})$
3: Train a node embedding algorithm using $\mathcal{G}_{obs}$ and $\mathbf{X}$ to obtain $\boldsymbol{z}_i$ for $1 \leqslant i \leqslant N$. Compute $\mathbf{D}_1$ using eq. (4.24).
4: Train a base classifier using $\mathcal{G}_{obs}$, $\mathbf{X}$ and $\mathbf{Y}_{\mathcal{L}}$ to obtain $\hat{c}_i$ for $1 \leqslant i \leqslant N$. Compute $\mathbf{D}_2$ using eq. (4.25).
5: Compute $\mathbf{D}$ using eq. (4.23).
6: Solve the optimization problem in (4.20) to obtain $A_{\widehat{\mathcal{G}}}$ (equivalently, $\widehat{\mathcal{G}}$).
7: **for** $s = 1$ **to** S **do**
8: Sample weights $\mathbf{W}_s$ using MC dropout by training a GCN over the graph $\widehat{\mathcal{G}}$.
9: **end for**
10: Approximate $p(\mathbf{Z}|\mathbf{Y}_{\mathcal{L}}, \mathbf{X}, \mathcal{G}_{obs})$ using eq. (4.22).

4.5.2 Link Prediction

In this setting, some of the links in $\mathcal{G}_{obs}$ are hidden or unobserved. The task is to predict the unseen links based on the knowledge of the (partially) observed $\mathcal{G}_{obs}$ and the node features $\mathbf{X}$. Thus in this case, the additional data beyond the graph is $\mathcal{D} = \mathbf{X}$.

In existing works, the link prediction problem is addressed by building deep learning based

generative models for graphs. In particular, various architectures of *variational graph autoencoders* (VGAEs) [39, 40, 213] aim to learn the posterior distribution of the node embedding $\boldsymbol{Z}$ conditioned on the observed graph $\mathcal{G}_{obs}$ and the node features $\mathbf{X}$. The inference model (encoder) often uses simplifying assumptions (e.g. mean-field approximation over nodes, diagonal covariance matrices for each node embedding) for the parametric form of the approximate variational posterior distribution $q(\boldsymbol{Z}|\mathcal{G}_{obs}, \mathbf{X})$. Various GNN architectures are used to learn the parameters of the model. The decoder is another deep learning model which explains how the graph is generated from the embeddings, i.e., it parameterizes $p(\mathcal{G}_{obs}|\boldsymbol{Z}, \mathbf{X})$. Typically the probability of a link in these models is dependent on the similarity of the embedding of the two incident nodes. Assuming a suitable prior $p(\boldsymbol{Z})$, the encoder and decoder is trained jointly to minimize the KL divergence between $q(\boldsymbol{Z}|\mathcal{G}_{obs}, \mathbf{X})$ and the true posterior $p(\boldsymbol{Z}|\mathcal{G}_{obs}, \mathbf{X})$. The learned embeddings are evaluated based on an amortized link prediction task for the unseen portion of the graph.

By contrast, we consider a Bayesian formulation, where we conduct Bayesian inference of the graph $\mathcal{G}$ in the encoder. Let us introduce a function $\mathcal{J}(\mathcal{G}, \mathcal{G}_{obs})$ that returns a graph such that the unobserved entries of the adjacency matrix of $\mathcal{G}_{obs}$ are replaced by the corresponding entries of $\mathcal{G}$. We then model the inference distribution as follows:

$$\begin{aligned} q(\boldsymbol{Z}|\mathcal{G}_{obs}, \mathbf{X}) &= \int q(\boldsymbol{Z}|\mathcal{J}(\mathcal{G}, \mathcal{G}_{obs}), \mathbf{X}) p(\mathcal{G}|\mathcal{G}_{obs}, \mathbf{X}) d\mathcal{G} \,, \\ &\approx q(\boldsymbol{Z}|\mathcal{J}(\widehat{\mathcal{G}}, \mathcal{G}_{obs}), \mathbf{X}) \,, \end{aligned} \tag{4.26}$$

where $\widehat{\mathcal{G}} = \arg\max_{\mathcal{G}} \; p(\mathcal{G}|\mathcal{G}_{obs}, \mathbf{X})$ is the MAP estimate from the non-parametric model. The intuitive idea behind this modeling is that if the non-parametric inference provides a reasonable approximation of the unobserved adjacency matrix entries, then an autoencoder trained on a graph that incorporates these approximate entries should learn better embeddings. For the graph learning step, we form the distance matrix $\mathbf{D}$ using the output of a graph autoencoder model as follows:

$$D_{ij}(\mathbf{X}, \mathcal{G}_{obs}) = \|\mathbb{E}_q[\boldsymbol{z}_i] - \mathbb{E}_q[\boldsymbol{z}_j]\|^2 \,. \tag{4.27}$$

The resulting Bayesian VGAE algorithm is summarized in Algorithm 4.3.

Algorithm 4.3 Bayesian VGAE

1: **Input:** $\mathcal{G}_{obs}$, $\mathbf{X}$
2: **Output:** $q(\boldsymbol{Z}|\mathcal{G}_{obs}, \mathbf{X})$
3: Train a node embedding algorithm using $\mathcal{G}_{obs}$ and $\mathbf{X}$ to obtain $q(\boldsymbol{z}_i|\mathcal{G}_{obs}, \mathbf{X})$ for $1 \leqslant i \leqslant N$.
4: Compute $\mathbf{D}$ using eq. (4.27).
5: Solve the optimization problem in (4.20) to obtain $A_{\widehat{\mathcal{G}}}$ (equivalently, $\widehat{\mathcal{G}}$).
6: Build a new graph $\mathcal{J}(\widehat{\mathcal{G}}, \mathcal{G}_{obs})$ and train the autoencoder on it to obtain $q(\boldsymbol{Z}|\mathcal{G}_{obs}, \mathbf{X})$ using eq. (4.26).

4.6 Numerical Experiments and Results

We evaluate the performance of the proposed BGCN (MMSBM) and BGCN (NP) algorithms in a transductive, semi-supervised node classification task. Additionally, we test the robustness of the BGCN (MMSBM) approach against an adversarial attack. Finally, we conduct a link prediction experiment by incorporating the non-parametric graph learning approach into several graph autoencoder models.

4.6.1 Datasets

We conduct experiments on benchmark citation network datasets: Cora [329], Citeseer [329], and Pubmed [330]. In these datasets, each node represents a research article and is associated with a bag-of-words feature vector derived from the keywords. Edges are formed whenever one document cites another. The direction of the citation is ignored and an undirected graph with a symmetric adjacency matrix is constructed. The node labels indicate the primary research topics addressed in the articles. The statistics of the citation datasets are summarized in Table 4.1.

Table 4.1: Statistics of the benchmark citation datasets.

Dataset	No. classes	No. features	No. nodes	No. Edges	Edge Density
Cora	7	1,433	2,485	5,069	0.04%
Citeseer	6	3,703	2,110	3,668	0.04%
Pubmed	3	500	19,717	44,324	0.01%

4.6.2 Semi-Supervised Node Classification

In this setting, we assume that we have access to several node labels per class and the goal is to predict the labels of the rest of the nodes. We consider three different experimental scenarios where we have 5, 10 and 20 labeled nodes per class in the training set. The data is split into train and test sets in two different ways. The first is the fixed data split with 20 training labels per class originating from [331], which has been extensively adopted in subsequent works [37, 38, 146]. In 5 and 10 training labels per class cases, we construct the fixed split of the data by using the first 5 and 10 labels in the original partition of [331]. On these fixed data partitions, each algorithm is run 50 times using different random initializations of the learnable weights. The second type of split is random where the training and test sets are created at random for each run. This provides a more robust comparison of the model performance as the specific split of data can have a significant impact in the limited training labels case. For this random splitting, we conduct 50 trials, where each trial is carried out on a different random partition of the data with a different random initialization of the learnable model parameters. The randomness solely arises in the fixed split scenarios due to the random initialization of weights, whereas random split settings show higher variance due to the additional randomness induced by the split of data.

We compare the proposed BGCN (MMSBM) and BGCN (NP) algorithms with the ChebyNet [62], the GCN [37], the GAT [38], the DFNET [146] (for only Cora and Citeseer due to runtime considerations), the SBM-GCN [41] and the BGCN (Copy) [211]. The hyperparameters of GCN are the same for all of the experiments and are based on [37]. The GCN has 2 layers with 16 hidden units, the learning rate is set to 0.01, the ℓ_2 regularization parameter is 0.0005, and the dropout rate is 50% at each layer. These hyperparameters are also used in the BGCN algorithms. In addition, the hyperparameters associated with MMSBM inference are set as follows: $\eta = 1$, $\alpha = 1$, $\rho = 0.001$, $n = 500$, $\epsilon_0 = 1$, $\tau = 1024$, and $\kappa = 0.5$. The hyperparameters of the non-parametric graph model are set using the heuristic in [180], such that the inferred graph ($\widehat{\mathcal{G}}$) is approximately 10 times denser compared to the observed graph ($\mathcal{G}_{obs}$). The hyperparameters of the other baseline models are borrowed from the original papers. The implementation of the GAT method, provided by the authors, employs a validation set of 500 node labels to monitor validation accuracy during training. The model that yields the minimum validation error is

selected as final model and is used for computing test set accuracy. We report results without this validation set monitoring as large validation sets are not always available and the other methods examined here do not require one.

We report the average classification accuracies along with their standard errors for the random partition scenario in Table 4.2. The results for the fixed split case are qualitatively similar and are deferred to Appendix B for conciseness. In order to highlight the advantage of graph modelling in the BGCN models, we conduct a Wilcoxon signed rank test to determine whether the BGCN algorithms are significantly better than the GCN. Statistically significant improvements of the BGCN models over the GCN at the 5% level are marked with asterisks.

We observe that the GCN outperforms the ChebyNet in most cases, which indicates that higher order graph convolutions are not advantageous for these datasets. Although the GAT requires much higher training time compared to the GCN, the incorporation of the attention mechanism is not particularly beneficial for these datasets. The DFNET-ATT model, which employs a complex, higher order convolution using feedback loop filters shows comparable performance to the GCN in most cases. The SBM-GCN approach targets joint learning of the node labels and a likely graph structure using an SBM. However, its performance is poor in scarce training data settings, possibly due to the ineffectiveness of the variational inference procedure employed for approximating the posterior of the graph. Motivated by the homophily property, present in real-world graph datasets, the recently proposed BGCN (Copy) model uses a simple node-copying based generative procedure for computationally efficient graph sampling and performs comparably to the other BGCN approaches in most cases.

The results illustrate the improvement in classification accuracy provided by the proposed BGCN (MMSBM) algorithm for Cora and Citeseer datasets. The improvement compared to the GCN is more pronounced when the number of available labels is limited to 10 or 5. In addition to increased accuracy, the BGCN (MMSBM) provides lower variance results in most tested scenarios. For the Pubmed dataset, the BGCN (MMSBM) has higher accuracy compared to the GCN for the 5-label case, but it is outperformed by other techniques for the 10 and 20-label cases. The Pubmed dataset has a much lower intra-community density than the other datasets and a heavy-tailed degree distribution. The assortative MMSBM is

Table 4.2: Accuracy (in %) of semi-supervised node classification on random splits. The best and the second best results in each column are shown in bold and marked with underline respectively. Higher numbers are better.

	Algorithms	5 labels	10 labels	20 labels
Cora	ChebyNet	61.7±6.8	72.5±3.4	78.8±1.6
	GCN	70.0±3.7	76.0±2.2	79.8±1.8
	GAT	70.4±3.7	76.6±2.8	79.9±1.8
	DFNET-ATT	72.3±2.9	75.8±1.7	79.3±1.8
	SBM-GCN	46.0±19	74.4±10	**82.6±0.2**
	BGCN (Copy)	73.8±2.7*	**77.6±2.6***	80.3±1.6
	BGCN (MMSBM)	**74.6±2.8***	77.5±2.6*	80.2±1.5
	BGCN (NP)	74.2±2.8*	76.9±2.2*	78.8±1.7
Citeseer	ChebyNet	58.5±4.8	65.8±2.8	67.5±1.9
	GCN	58.5±4.7	65.4±2.6	67.8±2.3
	GAT	56.7±5.1	64.1±3.3	67.6±2.3
	DFNET-ATT	60.5±1.2	63.2 ±2.9	66.3±1.7
	SBM-GCN	24.5±7.3	43.3±12	66.1±5.7
	BGCN (Copy)	63.9±4.2*	68.5±2.3*	70.2±2.0*
	BGCN (MMSBM)	63.0±4.8*	69.9±2.3*	71.1±1.8*
	BGCN (NP)	**64.9±4.6***	**70.1±1.9***	**71.4±1.6***
Pubmed	ChebyNet	62.7±6.9	68.6±5.0	74.3±3.0
	GCN	69.7±4.5	73.9±3.4	77.5±2.5
	GAT	68.0±4.8	72.6±3.6	76.4±3.0
	SBM-GCN	59.0±10	67.8±6.9	74.6±4.5
	BGCN (Copy)	71.0±4.2*	**74.6±3.3***	77.5±2.4
	BGCN (MMSBM)	70.2±4.5	73.3±3.1	76.0±2.6
	BGCN (NP)	**71.1±4.4***	**74.6±3.6***	**77.6±2.9**

thus a relatively poor choice for modelling the observed graph, and this might be the reason which prevents the Bayesian approach from improving the prediction accuracy.

In order to provide some insight into the information available from the posterior inference of the MMSBM parameters, we examine the 50 observed edges with lowest average posterior probability for both the Cora and Citeseer graphs. In the majority of cases the identified edges are inter-community (connecting edges with different labels) or have one node with very low degree (lower than 2). This accounts for 39 of the 50 edges for Cora and 42 of the 50 edges for Citeseer. For the unobserved edges, we analyze the most probable edges from the posterior. Most of these are intra-community edges (connecting nodes with the same label). For Cora, 177 of the 200 unobserved edges identified as most probable are intra-community, and for Citeseer, 197 intra-community edges are detected out of 200 most probable unobserved edges.

The proposed BGCN (NP) algorithm achieves either higher or competitive accuracies in most cases. The relative improvement compared to the GCN is more significant if the labelled data is scarce. Comparison with the BGCN (MMSBM) demonstrates that better or comparable accuracies can be achieved from this model, even if it does not target modelling the community structure of the graph explicitly. From Figure 4.1, we observe that in most cases, for the Cora and the Citeseer datasets, the proposed BGCN (NP) algorithm corrects more misclassifications of the GCN for low degree nodes. The same trend is observed for the Pubmed dataset. The empirical success of the GCN is primarily due to aggregating information with neighbors. As the low degree nodes have less opportunity to aggregate, performance is worse at these nodes. The proposed BGCN (NP) approach generates many additional links between similar nodes (Figure 4.2). This improves learning, particularly at low degree nodes.

In Figure 4.2, we compare the adjacency matrix ($A_{\widehat{\mathcal{G}}}$) of the MAP estimate graph $\widehat{\mathcal{G}}$ with the observed adjacency matrix $A_{\mathcal{G}_{obs}}$ for the Cora dataset. This reveals that compared to $A_{\mathcal{G}_{obs}}$, $A_{\widehat{\mathcal{G}}}$ has denser connectivity among the nodes with the same label. This provides a rationale of why the proposed BGCN (NP) outperforms the GCN in most cases.

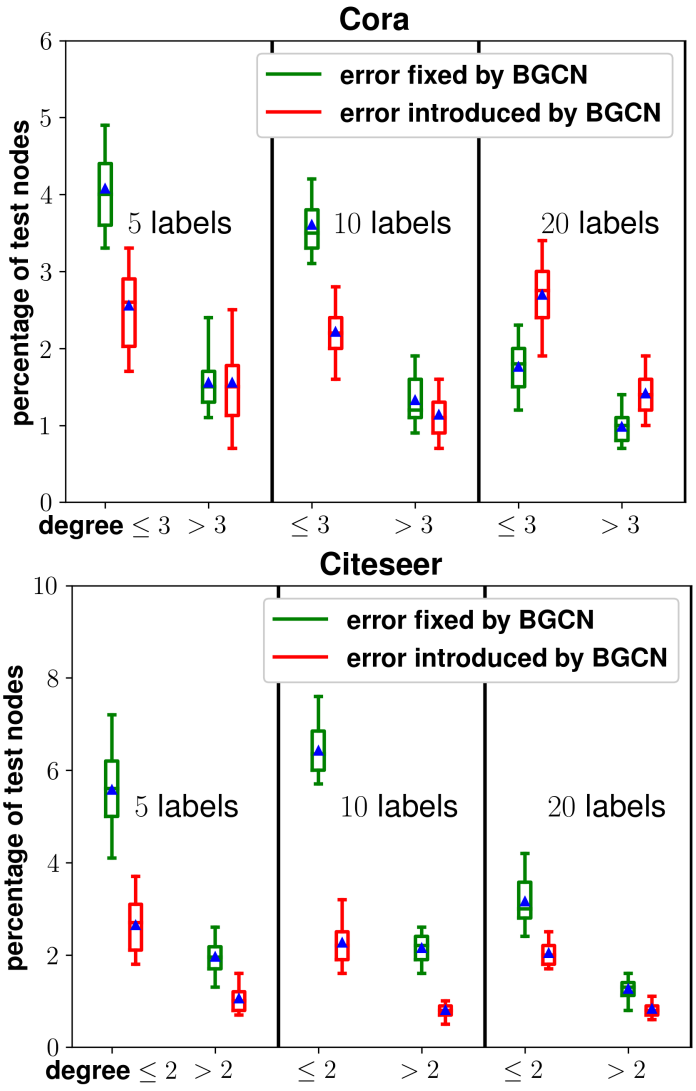


Figure 4.1: Boxplots of different categories of nodes in the Cora and Citeseer datasets based on the classification results of the GCN and the proposed BGCN (NP) algorithm. The two groups are formed by thresholding the degree of the nodes in the test set at the median value.

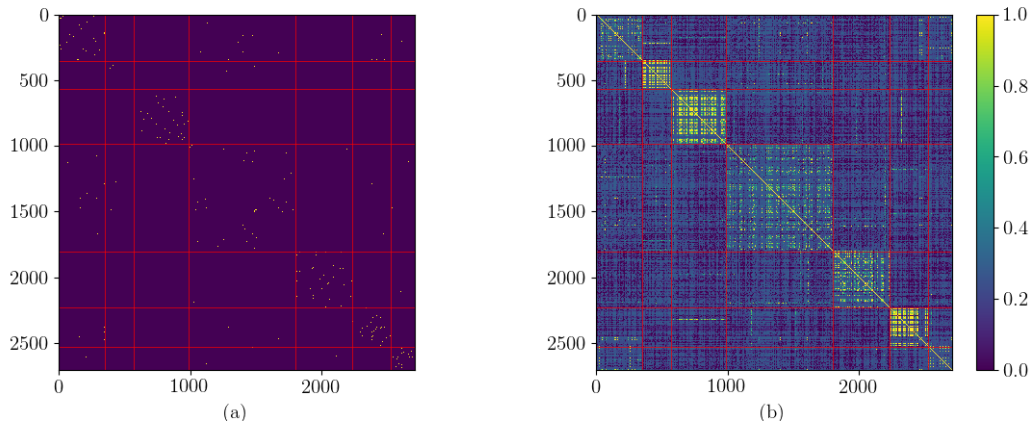


Figure 4.2: (a) the observed adjacency matrix ($A_{\mathcal{G}_{obs}}$) and (b) the MAP estimate of adjacency matrix ($A_{\widehat{\mathcal{G}}}$) from the non-parametric model for the Cora dataset. The node are reordered based on labels. The red lines show the class boundaries.

4.6.3 Node Classification under Adversarial Attack

Several studies have shown the vulnerability of deep neural networks to adversarial examples [332]. An adversarial attack with limited perturbation of the input graph is introduced in [333], which aims to demonstrate the vulnerability of the graph-based learning algorithms. Motivated by this study, we use a random attack mechanism to compare the robustness of the GCN and the BGCN (MMSBM) algorithms in the presence of noisy edges.

In each experiment, we target one node to attack. We choose a fixed number of perturbations $\Delta = d_{v_0}+2$, where d_{v_0} is the degree of a target node v_0. The random attack involves removing $\lceil (d_{v_0}+2)/2 \rceil$ nodes from the target node's set of neighbors, and adding $\lceil (d_{v_0}+2)/2 \rceil$ cross-community edges (randomly adding neighbors that have different labels than the target node) to the target node. For each target node, this procedure is repeated 5 times so that five perturbed graphs are generated. There are two types of adversarial attack mechanisms in [333]. In the first type, called an evasion attack, data is modified to fool an already trained classifier, and in the second, called a poisoning attack, the perturbation occurs before the model training. All of our experiments in this section are examples of the poisoning attack.

Similar to the setup in [333], we choose 40 nodes from the test set that are correctly classified and simulate attack on these nodes. The margin of classification for node v is defined as:

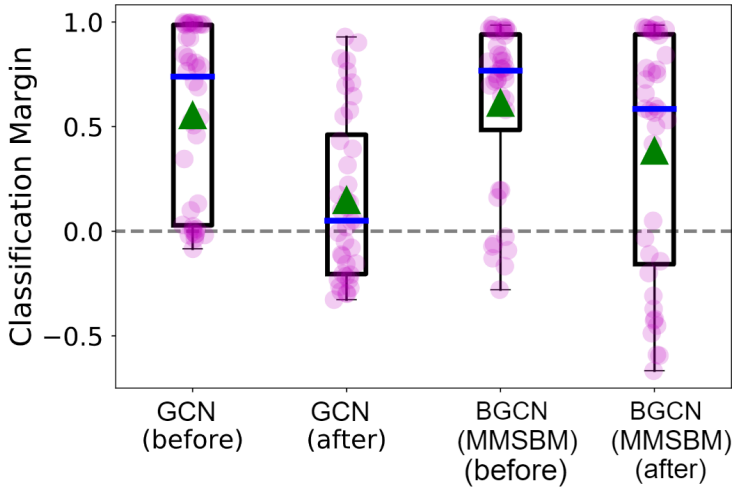
$$\text{margin}_v = \text{score}_v(c_{true}) - \max_{c \neq c_{true}} \text{score}_v(c)\,, \tag{4.28}$$

where c_{true} is the true class of node v and score_v denotes the softmax classification score vector reported by the classifier for node v. A correct classification leads to a positive margin; incorrect classifications are associated with negative margins. For each algorithm we choose the 10 nodes with the highest margin of classification and 10 nodes with the lowest positive margin of classification. The remaining 20 nodes are selected at random from the set of nodes correctly classified by both algorithms. Thus, among the 40 target nodes, the two algorithms are sharing at least 20 common nodes.

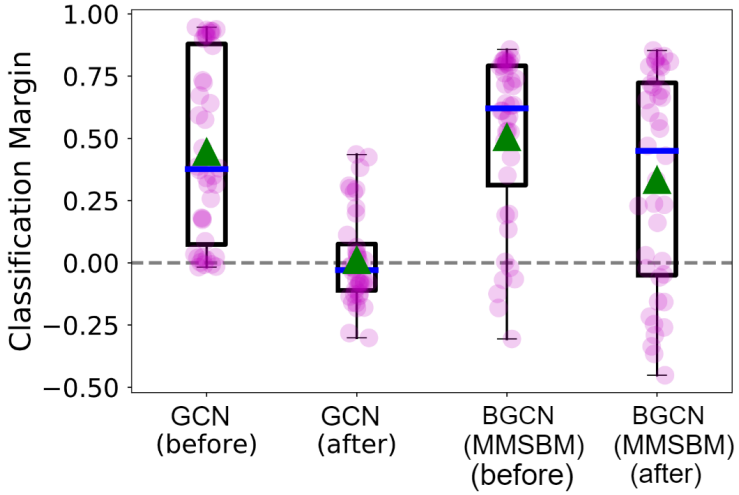
For each targeted node, we run the algorithm for 5 trials, where each trial corresponds to a different attacked graph and a different random initialization of the GCN and BGCN (MMSBM) weights. The average classification accuracy and margin with and without the random attack are reported in Table 4.3 and are computed using 40 selected target nodes and conducting 5 runs of the algorithms for each target. The accuracies in Table 4.3 are different from the ones in Table 4.2, since we report the accuracy for the 40 selected target nodes in this experiment, whereas results in Table 4.2 are accuracies on the entire test set.

Overall, the attack affects both algorithms severely. The GCN loses 30% in prediction accuracy for the Cora dataset and 44.5% for Citeseer whereas the decrease in prediction accuracy is more limited for the BGCN (MMSBM) algorithm with 17% for Cora and 20.5% for the Citeseer dataset. The BGCN (MMSBM) is able to maintain the classifier margin much better compared to the GCN. For the Citeseer dataset, the random attack almost eliminates the GCN's margin whereas the BGCN (MMSBM) suffers a 34% decrease, but retains a positive margin on average.

Figure 4.3 provides further insight concerning the impact of the attack on the two algorithms by depicting the distribution of average classifier margins over the targeted nodes before and after the random attack. Each circle in the figure shows the margin for one target node averaged over the 5 random perturbations of the graph. Note that some of the nodes have a negative margin prior to the random attack because we select the correctly classified nodes with lowest average margin based on 10 random trials and then perform another 5 random



(a)



(b)

Figure 4.3: Boxplots of the average classification margin for 40 nodes before and after random attack for the GCN and the BGCN (MMSBM) on (a) Cora and (b) Citeseer datasets. The box indicates 25-75 percentiles; the triangle represents the mean value; and the median is shown by a horizontal line. Whiskers extend to the minimum and maximum of data points.

	No attack	Random attack
Cora	Accuracy	
GCN	85.55%	55.50%
BGCN (MMSBM)	86.50%	69.50%
	Classifier Margin	
GCN	0.557	0.152
BGCN (MMSBM)	0.616	0.387
Citeseer	Accuracy	
GCN	88.5%	43.0%
BGCN (MMSBM)	87.0%	66.5%
	Classifier Margin	
GCN	0.448	0.014
BGCN (MMSBM)	0.507	0.335

Table 4.3: Accuracy and classifier margin for the no attack and random attack scenarios on Cora and Citeseer datasets.

trials using different initialization of the GCN and BGCN (MMSBM) weights to report the margins before attack. We observe that the attack causes nearly half of the target nodes to be wrongly classified for the GCN, whereas there are considerably fewer prediction changes for the BGCN (MMSBM) algorithm.

4.6.4 Link Prediction

We consider a link prediction task to demonstrate the usefulness of the learned embeddings from the Bayesian autoencoders using the non-parametric graph inference approach. As in [39], we split the links in 85/5/10% for training, validation and testing respectively. The validation and test sets contain the same number of non-links as links. During model training, the links in the validation and test sets are hidden while the node features are unaltered. We compare the Bayesian approach with the GAE and VGAE [39], the GRAPHITE-AE and VAE [40], and the DGLFRM [213] models. The hyperparameters of these baseline algorithms are selected according to the corresponding papers. Other common baselines, e.g., spectral clustering [334], Deepwalk [335], and node2vec [296], are not included since it has been demonstrated in [39] that the baselines we include significantly outperform them.

Table 4.4: Area Under the ROC Curve (AUC) and Average Precision (AP) (in %) for link prediction.

Algorithm	Cora	Citeseer	Pubmed
		AUC	
GAE	91.5±0.9	89.4±1.5	96.2±0.2
BGAE	91.8±0.8*	89.6±1.6*	96.2±0.2
VGAE	91.8±0.9	90.7±1.0	94.5±0.7
BVGAE	92.2±0.8*	91.2±1.0*	94.4±0.7
Graphite-AE	92.0±0.9	90.8±1.1	96.0±0.4
BGraphite-AE	92.4±0.9*	91.1±1.1*	96.0±0.4
Graphite-VAE	92.3±0.8	90.9±1.1	95.2±0.4
BGraphite-VAE	92.7±0.8*	91.4±1.1*	95.2±0.4
DGLFRM	93.1±0.6	93.9±0.7	95.9±0.1
BDGLFRM	93.2±0.6*	94.1±0.7*	95.9±0.2
		AP	
GAE	92.6±0.9	90.0±1.7	96.3±0.3
BGAE	92.8±0.9*	90.2±1.7*	96.3±0.2
VGAE	92.9±0.7	92.0±1.0	94.7±0.6
BVGAE	93.3±0.7*	92.5±1.0*	94.6±0.6
Graphite-AE	92.8±0.9	91.6±1.1	96.0±0.4
BGraphite-AE	93.1±0.9*	92.0±1.1*	96.0±0.4
Graphite-VAE	93.3±0.7	92.1±1.0	95.3±0.4
BGraphite-VAE	93.7±0.7*	92.6±1.0*	95.3±0.4
DGLFRM	93.8±0.6	94.5±0.7	96.4±0.1
BDGLFRM	93.9±0.6*	94.7±0.7*	96.3±0.1

We incorporate the non-parametric graph inference technique in the existing autoencoders to build a Bayesian version of these algorithms.

The Area Under the ROC Curve (AUC) and the Average Precision (AP) score are used as performance metrics. Table 4.4 shows the mean AUC and AP, together with standard errors, based on 50 trials. Each trial corresponds to a random split of the graph and a random initialization of the learnable parameters. We conduct a Wilcoxon signed rank test to determine the statistical significance of the improvement compared to the corresponding base model. Results marked with asterisks indicate settings where the test declares a significance at the 5% level.

From the results in Table 4.4, we observe the proposed approach improves link prediction performance for the Cora and Citeseer datasets compared to the baseline autoencoder models. The improvement is small but consistent over all autoencoder architectures and almost all of the random trials. No improvement is observed for Pubmed. To examine this further, we conducted an experiment where the ground-truth for the test set was provided to the autoencoders. The performance does not change from the reported values; this suggests that the models have reached accuracy limits for the Pubmed dataset.

4.7 Summary

In this chapter, we have introduced Bayesian graph neural networks, which provide an approach for incorporating uncertain graph information into graph based learning tasks. We provide an example of the Bayesian framework using a parametric random graph model, called assortative mixed membership stochastic block model and explain how approximate inference can be performed using a combination of stochastic optimization (to obtain maximum a posteriori estimates of the random graph parameters) and approximate variational inference through Monte Carlo dropout (to sample weights from the Bayesian GCN). The resulting BGCN (MMSBM) algorithm offers improved accuracy in semi-supervised node classification, particularly for the case where the number of training labels is small. The BGCN (MMSBM) is considerably more resilient to attack than the GCN. In addition, we propose the use of non-parametric modelling and inference of graphs for various learning tasks. In the proposed model, a higher edge weight between two nodes is more likely if the nodes are close in terms of a distance metric. An appropriate distance

metric can be chosen depending on the learning task which results in flexible, task-specific design of learning algorithms. The proposed model is adapted to a Bayesian learning framework to improve performance over baseline algorithms for node classification and link prediction.

Chapter 5
RNN with Particle Flow

5.1 Introduction

Spatio-temporal forecasting has many applications in intelligent traffic management, computational biology and finance, wireless networks and demand forecasting. Inspired by the surge of novel learning methods for graph structured data, many deep learning based spatio-temporal forecasting techniques have been proposed recently [50, 51]. In addition to the temporal patterns present in the data, these approaches can effectively learn and exploit spatial relationships among the time-series using various Graph Neural Networks (GNNs) [37, 62]. Recent works establish that graph-based spatio-temporal models outperform the graph-agnostic baselines [50, 288]. In spite of their accuracy in providing point forecasts, these models have a serious drawback as they cannot gauge the uncertainty in their predictions. When decisions are made based on forecasts, the availability of a confidence or prediction interval can be vital.

There are numerous probabilistic forecasting techniques for multivariate time-series, for example, DeepAR [52], DeepState [54], DeepFactors [53], and the normalizing flow-based algorithms [266, 269]. Although these algorithms can characterize uncertainty via confidence intervals, they are not designed to incorporate side-knowledge provided in the form of a graph.

In this Chapter, we model multivariate time-series as random realizations from a nonlinear state-space model, and target Bayesian inference of the hidden states for probabilistic forecasting. The general framework we propose can be applied to univariate or multivariate forecasting problems, can incorporate additional covariates, can process an observed graph, and can be combined with data-adaptive graph learning procedures. For the concrete example algorithm deployed in experiments, we build the dynamics of the state-space model using graph convolutional recurrent architectures. We develop an inference procedure that employs particle flow, an alternative to particle filters, that can conduct more effective inference for high-dimensional states.

The novel contributions in this chapter are as follows: a) we propose a graph-aware

stochastic recurrent network architecture and inference procedure that combine graph convolutional learning, a probabilistic state-space model, and particle flow; b) we demonstrate via experiments on graph-based traffic datasets that a specific instantiation of the proposed framework can provide point forecasts that are as accurate as the state-of-the-art deep learning based spatio-temporal models. The prediction error is also comparable to the existing deep learning based techniques for benchmark non-graph multivariate time-series datasets; and c) we show that the proposed method provides a superior characterization of the prediction uncertainty compared to existing probabilistic multivariate time-series forecasting methods, both for datasets where a graph is available and for settings where no graph is available.

The rest of the chapter is organized as follows. Section 5.2 describes the forecasting problem we address. The proposed approach is detailed in Section 5.3. Experimental results on several real-world datasets are reported and discussed in Section 5.4. We provide a summary of this chapter in Section 5.5.

5.2 Problem Statement

We address the task of discrete-time multivariate time-series prediction, with the goal of forecasting multiple time-steps ahead. We assume that there is access to a historical dataset for training, but after training the model must perform prediction based on a limited window of historical data. Let $\mathbf{y}_t \in \mathbb{R}^{N\times 1}$ be an observed multivariate signal at time t and $\mathbf{Z}_t \in \mathbb{R}^{N\times d_z}$ be an associated set of covariates. The i-th element of $\mathbf{y}_t$ is the observation associated with time-series i at time-step t.

We also allow for the possibility that there is access to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of N nodes and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ denotes the set of edges. In this case, each node corresponds to one time-series. The edges indicate probable predictive relationships between the variables, i.e., the presence of an edge (i, j) suggests that the historical data for time-series i is likely to be useful in predicting time-series j. The graph may be directed or undirected.

The goal is to construct a model that is capable of processing, for some time offset t_0, the data $\mathbf{Y}_{t_0+1:t_0+P}$, $\mathbf{Z}_{t_0+1:t_0+P+Q}$ and (possibly) the graph $\mathcal{G}$, to estimate $\mathbf{Y}_{t_0+P+1:t_0+P+Q}$. The prediction algorithm should produce both point estimates and prediction intervals. The

performance metrics for the point estimates include mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean squared error (RMSE). For the prediction intervals, the performance metrics include the Continuous Ranked Probability Score (CRPS) [336], and the P10, P50, and P90 Quantile Losses (QL) [52, 53]. Expressions for these performance metrics are provided in Section 5.4.2.

5.3 Methodology

5.3.1 State-Space Model

We postulate that $\mathbf{y}_t \in \mathbb{R}^{N\times 1}$ is the observation from a Markovian state space model with hidden state $\mathbf{X}_t \in \mathbb{R}^{N\times d_x}$. We denote by $\mathbf{x}_t$ and $\mathbf{z}_t$ the vectorizations of $\mathbf{X}_t$ and $\mathbf{Z}_t$, respectively. The state space model is:

$$\mathbf{x}_1 \sim p_1(\cdot, \mathbf{z}_1, \rho)\,, \tag{5.1}$$

$$\mathbf{x}_t = g_{\mathcal{G},\psi}(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t, \mathbf{v}_t),\ \text{for } t > 1\,, \tag{5.2}$$

$$\mathbf{y}_t = h_{\mathcal{G},\phi}(\mathbf{x}_t, \mathbf{z}_t, \mathbf{w}_t),\ \text{for } t \geqslant 1\,. \tag{5.3}$$

Here $\mathbf{v}_t \sim p_v(\cdot|\mathbf{x}_{t-1}, \sigma)$ and $\mathbf{w}_t \sim p_w(\cdot|\mathbf{x}_t, \gamma)$ are the noises in the dynamic and measurement models respectively. ρ, σ and γ are the parameters of the distribution of the initial state $\mathbf{x}_1$, process noise $\mathbf{v}_t$ and measurement noise $\mathbf{w}_t$ respectively. g and h denote the state transition and measurement functions, possibly linear or nonlinear, with parameters ψ and ϕ respectively. The subscript $\mathcal{G}$ in g and h indicates that the functions are potentially dependent on the graph topology. We assume that $h_{\mathcal{G},\phi}(\mathbf{x}_t, \mathbf{z}_t, 0)$ is a $\mathcal{C}^1$ function in $\mathbf{x}_t$, i.e., $h_{\mathcal{G},\phi}(\mathbf{x}_t, \mathbf{z}_t, \mathbf{0})$ is a differentiable function whose first derivative with respect to $\mathbf{x}_t$ is continuous. The complete set of the unknown parameters is formed as: $\Theta = \{\rho, \psi, \sigma, \phi, \gamma\}$. Figure 5.1 depicts the graphical model relating the observed variables ($\mathbf{y}_t$ and $\mathbf{z}_t$) to the latent variables ($\mathbf{v}_t$, $\mathbf{w}_t$) and the graph ($\mathcal{G}$).

With the proposed formulation, we can modify recurrent graph convolutional architectures when designing the function g. When a meaningful graph is available, such architectures significantly outperform models that ignore the graph. For example, we conduct experiments by incorporating into our general model the Adaptive Graph Convolutional Gated Recurrent

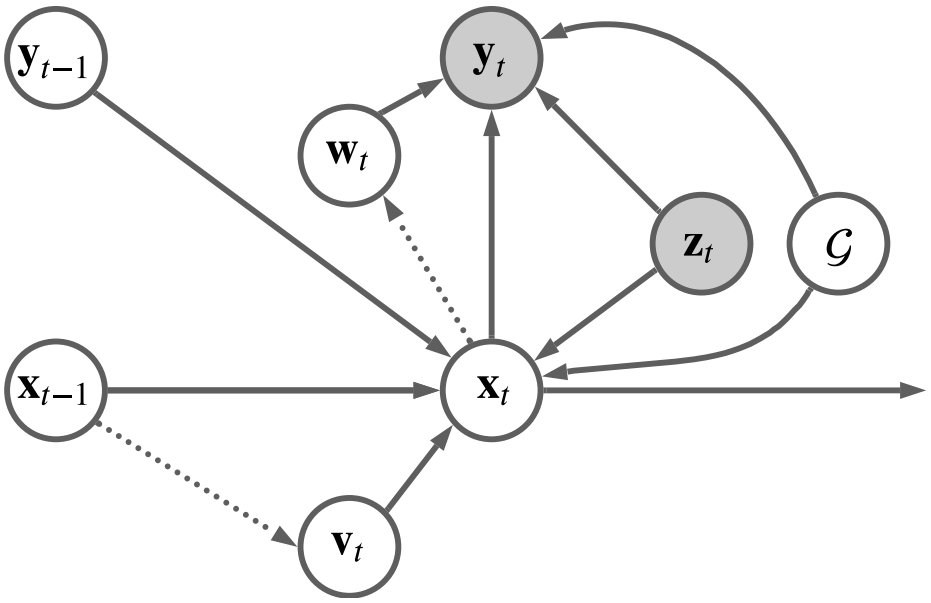


Figure 5.1: The graphical model representation of the state-space model in Section 5.3.1

Units (AGCGRU) presented in [51]. The AGCGRU combines (i) a module that adapts the provided graph based on observed data, (ii) graph convolution to capture spatial relations, and (iii) a GRU to capture evolution in time. The example model used for experiments thus employs an L-layer AGCRU with additive Gaussian noise to model the system dynamics g:

$$\mathbf{x}_t = AGCGRU^{(L)}_{\mathcal{G},\psi}(\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t) + \mathbf{v}_t\,, \tag{5.4}$$

$$\mathbf{y}_t = \mathbf{W}_\phi \mathbf{x}_t + \mathbf{w}_t\,. \tag{5.5}$$

In this model, we have $p_v(\mathbf{v}_t) = \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, i.e., the latent variables for the dynamics are independent. The initial state distribution is also chosen to be isotropic Gaussian, i.e., $p_1(\mathbf{x}_1, \mathbf{z}_1, \rho) = \mathcal{N}(\mathbf{0}, \rho^2\mathbf{I})$. The parameters ρ and σ are learnable variance parameters. The observation model g incorporates a linear projection matrix $\mathbf{W}_\phi$. The latent variable $\boldsymbol{w}_t$ for the emission model is modelled as Gaussian with variance dependent on $\mathbf{x}_t$ via a learnable softplus function:

$$p_w(\mathbf{w}_t|\mathbf{x}_t) = \mathcal{N}\Big(\mathbf{0}, \mathrm{diag}\Big(\mathrm{softplus}(\boldsymbol{C}_\gamma \boldsymbol{x}_t)\Big)^2\Big)\,. \tag{5.6}$$

5.3.2 Inference

We assume that a dataset $\mathcal{D}_{trn}$ is available for training. Although this data may be derived from a single time-series, because our task is to predict $\mathbf{y}_{t_0+P+1:t_0+P+Q}$ using a limited historical window $\mathbf{y}_{t_0+1:t_0+P}$, we splice the time-series and thus construct multiple training examples, denoted by $(\mathbf{y}^{(m)}_{1:P}, \mathbf{y}^{(m)}_{P+1:P+Q})$. In the training set, all of these observations are available; in the test set $\mathbf{y}_{P+1:P+Q}$ are not. In addition, the associated covariates $\mathbf{z}_{1:P+Q}$ are known for both training and test sets.

Inference involves an iterative process. We randomly initialize the parameters of the model to obtain Θ_0. Subsequently, at the k-th iteration of the algorithm (processing the k-th training batch), we first draw samples $\{\mathbf{y}^i_{P+1:P+Q}\}_{i=1}^{N_p}$ from the distribution $p_{\Theta_{k-1}}(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q})$. With this set of samples, we can subsequently apply a gradient descent procedure to obtain the updated model parameters Θ_k. We discuss each of these steps in turn as follows.

Sampling

In a Bayesian setting with *known* model parameters $\Theta = \Theta_{k-1}$, we would aim to form a prediction by approximating the posterior predictive distribution of the forecasts, i.e., $p_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P})$. (For conciseness we drop the time-offset t_0).

$$p_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q}) = \int \prod_{t=P+1}^{P+Q} \Big(p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t, \mathbf{z}_t) p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t) \Big) \\ p_\Theta(\mathbf{x}_P|\mathbf{y}_{1:P}, \mathbf{z}_{1:P}) d\mathbf{x}_{P:P+Q} \,. \tag{5.7}$$

Since the integral in eq. (5.7) is analytically intractable for a general nonlinear state-space model, we take a Monte Carlo approach as follows:

Step 1: For $1 \leqslant t \leqslant P$, we apply an EDH [21] particle flow algorithm (details in Section 2.1.6) with N_p particles for the state-space model specified by eqs. (5.1), (5.2) and (5.3) to recursively approximate the posterior distribution of the states:

$$p_\Theta(\mathbf{x}_t|\mathbf{y}_{1:t}, \mathbf{z}_{1:t}) \approx \frac{1}{N_p} \sum_{j=1}^{N_p} \delta_{\mathbf{x}^i_t}(\mathbf{x}_t) \,. \tag{5.8}$$

Here $\{\mathbf{x}_t^j\}_{j=1}^{N_p}$ are approximately distributed according to the posterior distribution of $\mathbf{x}_t$. The past trajectory of each sample $\mathbf{x}_{1:P-1}^j$ can be discarded since the proposed model only needs samples of $\mathbf{x}_P$ to construct the forecast.

Step 2: For $P+1 \leqslant t \leqslant P+Q$, we iterate between the following two steps:

a. We sample $\mathbf{x}_t^j$ from $p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}^j, \mathbf{y}_{t-1}^j, \mathbf{z}_t)$ (for $t > P+1$) or from $p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}^j, \mathbf{y}_{t-1}, \mathbf{z}_t)$ (for $t = P+1$) for $1 \leqslant j \leqslant N_p$. This amounts to a state transition at time t to obtain the current state $\mathbf{x}_t^j$ from the previous state $\mathbf{x}_{t-1}^j$, using eq. (5.2).

b. We sample $\mathbf{y}_t^j$ from $p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t^j, \mathbf{z}_t)$ for $1 \leqslant j \leqslant N_p$, i.e., we use $\mathbf{x}_t^j$ in the measurement model, specified by eq. (5.3), to sample $\mathbf{y}_t^j$.

A Monte Carlo (MC) approximation of the integral in eq. (5.7) is then formed as:

$$p_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q}) \approx \prod_{t=P+1}^{P+Q} \frac{1}{N_p} \sum_{j=1}^{N_p} \delta(\mathbf{y}_t - \mathbf{y}_t^j)\,. \tag{5.9}$$

Each $\mathbf{y}_{P+1:P+Q}^j$ is approximately distributed according to the joint posterior distribution of $\mathbf{y}_{P+1:P+Q}$. The resulting algorithm is summarized in Algorithm 5.1. A block diagram of the probabilistic forecasting procedure is shown in Figure 5.2.

Parameter Update

With the predictive samples $\{\mathbf{y}_{P+1:P+Q}^j\}_{j=1}^{N_p}$, we can update the model parameters via Stochastic Gradient Descent (SGD) to obtain $\Theta = \Theta_k$.

If our focus is on obtaining a point estimate of the forecast, then we can perform optimization on the training set with respect to a loss function derived from Mean Absolute Error (MAE) or Mean Square Error (MSE). The point forecast $\hat{\mathbf{y}}_{P+1:P+Q}^{(m)}$ is obtained based on a statistic such as the mean or median of the samples $\{\mathbf{y}_{P+1:P+Q}^{j,(m)}\}_{j=1}^{N_p}$. The MAE loss function on a dataset indexed by $\mathcal{D}$ can then be expressed as:

$$\mathcal{L}_{\text{MAE}}(\Theta, \mathcal{D}) = \frac{1}{NQ|\mathcal{D}|} \sum_{m \in \mathcal{D}} \sum_{t=P+1}^{P+Q} \left\| \mathbf{y}_t^{(m)} - \hat{\mathbf{y}}_t^{(m)} \right\|_1 . \tag{5.10}$$

In an alternate approach, we could consider the maximization of the marginal log-likelihood

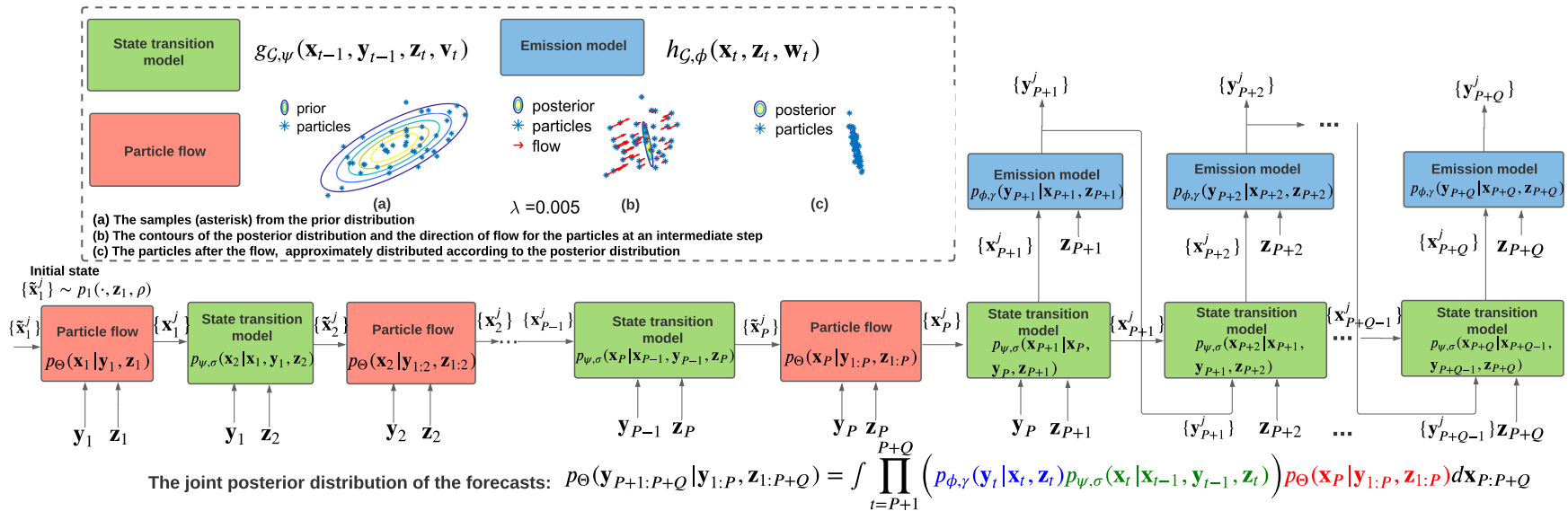


Figure 5.2: Probabilistic forecasting from the state-space model using particle flow. Migration of particles from a 2-d Gaussian prior to a 2-d Gaussian posterior distribution is illustrated as an example.

Algorithm 5.1 Sequence to sequence prediction

Input: $\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q}$, and Θ
Output: $\{\mathbf{y}^j_{P+1:P+Q}\}_{j=1}^{N_p}$
Initialization: Sample $\eta_0^j \sim p_1(\mathbf{x}_1, \mathbf{z}_1, \rho)$, $j = 1 : N_p$.
for $t = 1, 2, ..., P$ **do**
  if $t > 1$ **then**
    Sample $\eta_0^j \sim p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}^j_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t)$, $j = 1 : N_p$ as: $\eta_0^j = g_{\mathcal{G},\psi}(\mathbf{x}^j_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t, \mathbf{v}_t)$.
  end if
  Use EDH particle flow (Algorithm 2.3) to obtain $\{\eta_1^j\}_{j=1}^{N_p}$ from $\{\eta_0^j\}_{j=1}^{N_p}$, $\mathbf{z}_t$, and $\mathbf{y}_t$.
  Set $\mathbf{x}_t^j = \eta_1^j$.
end for
for $t = P + 1, P + 2, ..., P + Q$ **do**
  if $t = P + 1$ **then**
    Sample $\mathbf{x}^j_{P+1} \sim p_{\psi,\sigma}(\mathbf{x}_{P+1}|\mathbf{x}^j_P, \mathbf{y}_P, \mathbf{z}_{P+1})$, $j = 1 : N_p$ as: $\mathbf{x}^j_{P+1} = g_{\mathcal{G},\psi}(\mathbf{x}^j_P, \mathbf{y}_P, \mathbf{z}_{P+1}, \mathbf{v}_{P+1})$.
  else
    Sample $\mathbf{x}_t^j \sim p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}^j_{t-1}, \mathbf{y}^j_{t-1}, \mathbf{z}_t)$, $j = 1 : N_p$ as: $\mathbf{x}_t^j = g_{\mathcal{G},\psi}(\mathbf{x}^j_{t-1}, \mathbf{y}^j_{t-1}, \mathbf{z}_t, \mathbf{v}_t)$.
  end if
  Sample $\mathbf{y}_t^j \sim p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t^j, \mathbf{z}_t)$, $j = 1 : N_p$ as: $\mathbf{y}_t^j = h_{\mathcal{G},\phi}(\mathbf{x}_t^j, \mathbf{z}_t, \mathbf{w}_t)$.
end for
Form the Monte Carlo estimate using eq. (5.9).

over the training set. In that case, a suitable loss function is:

$$\mathcal{L}_{\text{prob}}(\Theta, \mathcal{D}) = -\frac{1}{|\mathcal{D}|} \sum_{m \in \mathcal{D}} \log p_\Theta(\mathbf{y}^{(m)}_{P+1:P+Q}|\mathbf{y}^{(m)}_{1:P}, \mathbf{z}^{(m)}_{1:P+Q}) \,, \tag{5.11}$$

where we approximate the predictive likelihood as:

$$\widehat{p}_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q}) = \prod_{t=P+1}^{P+Q} \left[\frac{1}{N_p} \sum_{j=1}^{N_p} p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t^j, \mathbf{z}_t) \right], \tag{5.12}$$

using eq. (5.7). This loss formulation is similar to the MC variational objectives in [225–227]. If we use the particle flow particle filter [26], then the sampled particles and the propagated forecasts form an unbiased approximation of the distribution $p_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q})$. By Jensen's inequality, the summation over the log terms in eq. (5.11) is thus a lower bound for the desired $\mathbb{E}[\log p_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q})]$ that converges to it as $N_p \to \infty$.

In each training mini-batch, for each training example, we perform a forward pass through

Algorithm 5.2 Model training and testing

Input: Training and test data: $\{\mathbf{y}_{1:P+Q}^{(m)}, \mathbf{z}_{1:P+Q}^{(m)}\}_{m\in\mathcal{D}_{trn}}$, $\{\mathbf{y}_{1:P}^{(n)}, \mathbf{z}_{1:P+Q}^{(n)}\}_{n\in\mathcal{D}_{test}}$
Output: $\{\hat{p}_{\hat{\Theta}}(\mathbf{y}_{P+1:P+Q}^{(n)}|\mathbf{y}_{1:P}^{(n)}, \mathbf{z}_{1:P+Q}^{(n)})\}_{n\in\mathcal{D}_{test}}$
Hyperparameters: Number of iterations N_{iter}, step-size $\{\zeta_k\}_{k=1}^{N_{iter}}$
Initialization: random initialization for the system parameters Θ_0.
Model training:
Set $k = 1$.
while $k \leqslant N_{iter}$ **do**
  Sample a minibatch $\mathcal{D} \subset \mathcal{D}_{trn}$.
  Compute the approximate posterior distribution of the forecasts $\{\hat{p}_{\Theta_{k-1}}(\mathbf{y}_{P+1:P+Q}^{(m)}|\mathbf{y}_{1:P}^{(m)}, \mathbf{z}_{1:P+Q}^{(m)})\}_{m\in\mathcal{D}}$ using Algorithm 5.1 with the current parameters Θ_{k-1}.
  Compute the gradient of the chosen loss function $\mathcal{L}(\Theta, \mathcal{D})$ with respect to model parameters Θ at Θ_{k-1}.
  Update the system parameters using SGD: $\Theta_k = \Theta_{k-1} - \zeta_k \nabla_\Theta \mathcal{L}(\Theta, \mathcal{D})\Big|_{\Theta=\Theta_{k-1}}$.
  $k = k + 1$.
end while
Save the estimated model $\widehat{\Theta} = \Theta_{N_{iter}}$.
Testing:
Compute the test set forecast posterior distributions $\{\hat{p}_{\hat{\Theta}}(\mathbf{y}_{P+1:P+Q}^{(n)}|\mathbf{y}_{1:P}^{(n)}, \mathbf{z}_{1:P+Q}^{(n)})\}_{n\in\mathcal{D}_{test}}$ using Algorithm 5.1 with the estimated model parameters $\widehat{\Theta}$.

the model using Algorithm 5.1 to obtain approximate forecast posteriors and then update all the model parameters using SGD via backpropagation. Algorithm 5.2 summarizes the learning of the model parameters Θ.

5.4 Numerical Experiments and Results

We perform several experiments on four graph-based and four non-graph based public datasets to evaluate the proposed methods.

5.4.1 Datasets

We evaluate our proposed algorithm on four publicly available traffic datasets, namely PeMSD3, PeMSD4, PeMSD7, and PeMSD8. These datasets are obtained from the Caltrans

Performance Measurement System (PeMS) [337] and have been used in multiple previous works [51, 276, 286, 292, 293]. Each of these datasets consists of traffic speed records, collected from loop detectors, and aggregated over 5 minute intervals, resulting in 288 data points per detector per day. The statistics of these PeMS datasets are summarized in Table 5.1. In the non-graph setting, we conduct experiments on the Electricity [338], Traffic [338], Taxi [265], and Wikipedia [265] datasets. The Electricity[1] dataset contains electricity consumption for 370 clients. The Traffic[2] dataset is composed of 963 time-series of lane occupancy rates in San Francisco. The Taxi[3] dataset contains counts of taxis on different roads of New York, and the Wikipedia[4] dataset specifies the number of daily clicks of 2000 web links. The detailed statistics of these datasets are provided in Table 5.2.

Table 5.1: Summary statistics of the PeMS road traffic datasets.

Dataset	PeMSD3	PeMSD4	PeMSD7	PeMSD8
No. nodes	358	307	228	170
No. time steps	26208	16992	12672	17856
Interval	5 min	5 min	5 min	5 min

Table 5.2: Summary statistics of the multivariate non-graph datasets.

Dataset	No. time series (N)	Domain	Freq.	No. time steps	Prediction length (Q)
Electricity	370	$\mathbb{R}^+$	Hourly	5833	24
Traffic	963	$(0, 1)$	Hourly	4001	24
Taxi	1214	$\mathbb{N}$	30 Minutes	1488	24
Wikipedia	2000	$\mathbb{N}$	Daily	792	30

5.4.2 Definitions of Evaluation Metrics

The point forecasts are evaluated by computing mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean squared error (RMSE). For the test-set indexed by $\mathcal{D}_{test}$, let $\mathbf{y}_t^{(m)} \in \mathbb{R}^N$ and $\hat{\mathbf{y}}_t^{(m)} \in \mathbb{R}^N$ denote the ground truth and the prediction at horizon t

[1] `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`
[2] `https://archive.ics.uci.edu/ml/datasets/PEMS-SF`
[3] `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`
[4] `https://github.com/mbohlkeschneider/gluon-ts/tree/mv_release/datasets`

for m-th test example respectively. The average MAE, MAPE, and RMSE at horizon t are defined as follows:

$$\text{MAE}(\mathcal{D}_{test}, t) = \frac{1}{N|\mathcal{D}_{test}|} \sum_{m \in \mathcal{D}_{test}} \left\| \mathbf{y}_t^{(m)} - \hat{\mathbf{y}}_t^{(m)} \right\|_1 , \tag{5.13}$$

$$\text{MAPE}(\mathcal{D}_{test}, t) = \frac{1}{N|\mathcal{D}_{test}|} \sum_{m \in \mathcal{D}_{test}} \sum_{i=1}^{N} \frac{\left| \mathbf{y}_{t,i}^{(m)} - \hat{\mathbf{y}}_{t,i}^{(m)} \right|}{|\mathbf{y}_{t,i}^{(m)}|} , \tag{5.14}$$

$$\text{RMSE}(\mathcal{D}_{test}, t) = \sqrt{\frac{1}{N|\mathcal{D}_{test}|} \sum_{m \in \mathcal{D}_{test}} \left\| \mathbf{y}_t^{(m)} - \hat{\mathbf{y}}_t^{(m)} \right\|_2^2} . \tag{5.15}$$

For comparison among the probabilistic forecasting models, we compute the Continuous Ranked Probability Score (CRPS) [336], and the P10 and P90 Quantile Losses (QL) [52,53]. Let $F(\cdot)$ be the Cumulative Distribution Function (CDF) of the forecast of the true value $x \in \mathbb{R}$. We denote by $\mathbf{1}(x \leqslant z)$ the indicator function that attains the value 1 if $x \leqslant z$ and the value 0 otherwise. The continuous ranked probability score (CRPS) is defined as:

$$\text{CRPS}(F, x) = \int_{-\infty}^{\infty} \Big(F(z) - \mathbf{1}\big(x \leqslant z\big) \Big)^2 dz \, . \tag{5.16}$$

CRPS is a proper scoring function, i.e., it attains its minimum value of zero when the forecast CDF F is a step function at the ground truth x. The average CRPS at horizon t is defined as the average marginal CRPS across different time-series.

$$\text{CRPS}_{avg}(\mathcal{D}_{test}, t) = \frac{1}{N|\mathcal{D}_{test}|} \sum_{m \in \mathcal{D}_{test}} \sum_{i=1}^{N} \text{CRPS}(F_{t,i}^{(m)}, \mathbf{y}_{t,i}^{(m)}) \, , \tag{5.17}$$

where $F_{t,i}^{(m)}(\cdot)$ is the marginal CDF of the forecast at horizon t for i-th time-series in m-th test example.

Let $F_{t,sum}^{(m)}(\cdot)$ be the CDF of the sum of the forecasts of all time-series at horizon t in the m-th test example. The (normalized) CRPS_{sum} is defined as:

$$\text{CRPS}_{sum}(\mathcal{D}_{test}) = \frac{\sum_t \sum_{m \in \mathcal{D}_{test}} \text{CRPS}(F_{t,sum}^{(m)}, \sum_{i=1}^{N} \mathbf{y}_{t,i}^{(m)})}{\sum_t \sum_{m \in \mathcal{D}_{test}} |\sum_{i=1}^{N} \mathbf{y}_{t,i}^{(m)}|} \, . \tag{5.18}$$

For a given quantile $\alpha \in (0, 1)$, a true value x, and an α-quantile prediction $\hat{x}(\alpha) = F^{-1}(\alpha)$,

the α-quantile loss is defined as:

$$\mathrm{QL}\Big(x, \hat{x}(\alpha)\Big) = 2\Bigg(\alpha\Big(x - \hat{x}(\alpha)\Big)\mathbf{1}\Big(x > \hat{x}(\alpha)\Big) + (1-\alpha)\Big(\hat{x}(\alpha) - x\Big)\mathbf{1}\Big(x \leqslant \hat{x}(\alpha)\Big)\Bigg). \quad (5.19)$$

The average (normalized) quantile loss (QL) is defined as follows:

$$\mathrm{QL}_{avg}(\mathcal{D}_{test}, t, \alpha) = \frac{\sum_{m\in\mathcal{D}_{test}}\sum_{i=1}^{N}\mathrm{QL}\Big(\mathbf{y}_{t,i}^{(m)}, \hat{\mathbf{y}}_{t,i}^{(m)}(\alpha)\Big)}{\sum_{m\in\mathcal{D}_{test}}\sum_{i=1}^{N}|\mathbf{y}_{t,i}^{(m)}|}. \quad (5.20)$$

The P10QL metric is obtained by setting $\alpha = 0.1$ in eq. (5.20); the P90QL metric corresponds to $\alpha = 0.9$ and the ND (P50QL) metric is obtained using $\alpha = 0.5$.

5.4.3 Experiments on PeMS datasets

Preprocessing

For the PeMS datasets, missing values are filled by the last known value in the same series. The training, validation and test split is set at 70/10/20% chronologically and standard normalization of the data is used as in [50]. We use one hour of historical data ($P = 12$) to predict the traffic for the next hour ($Q = 12$). Graphs associated with the datasets are constructed using the procedure in [286].

Baselines

To demonstrate the effectiveness of our approach, we compare the proposed AGCGRU+flow algorithm with four different classes of forecasting techniques, listed as follows[5]:

Graph agnostic statistical and machine learning based point forecasting models:

- HA (Historical Average) uses the seasonality of the historical data.
- ARIMA [47] (Auto-Regressive Integrated Moving Average) model is implemented using a Kalman filter.

[5] Some of the recent spatio-temporal models such as [277, 298, 339] do not have publicly available code. Although the codes for [288, 292, 340] are available, these works use different datasets for evaluation. We could not obtain sensible results from these models for our datasets, even with considerable hyperparameter tuning. The code for [266, 268] is not publicly available.

- VAR [230] (Vector Auto-Regressive) model is a generalization of AR model to multivariate setting.
- SVR [341] (Support Vector Regression) is implemented using a linear kernel.
- FNN (Feedforward Neural Network) uses a 2-layer architecture with ReLU activation function at the hidden layer.
- FC-LSTM [342] (Fully Connected Long Short Term Memory) is an encoder-decoder RNN architecture for sequence to sequence prediction using fully connected LSTM layers.

Spatio-temporal point forecast models:

- DCRNN [50] (Diffusion Convolutional Recurrent Neural Network) combines diffusion convolution with GRU to form an encoder-decoder architecture for sequence to sequence prediction.
- STGCN [276] (Spatio-Temporal Graph Convolutional Network) uses gated temporal convolution with graph convolution.
- ASTGCN [293] (Attention Spatial-Temporal Graph Convolutional Network) employs spatial and temporal attentions to learn recent and seasonal patterns.
- GWN [287] (Graph WaveNet) is built using graph convolution and dilated causal convolution, it also has a provision for learning a graph.
- GMAN [295] (Graph Multi-Attention Network) consists of multiple spatio-temporal attention blocks to form an encoder-decoder architecture with transform attention between encoder and decoder.
- AGCRN [51] (Adaptive Graph Convolutional Recurrent Network) is equipped with a node adaptive parameter learning for graph convolution using adaptive adjacency within a GRU.
- LSGCN [286] (Long Short-term Graph Convolutional Network) combines a novel attention mechanism and graph convolution, integrated into a spatial gated block.

Deep learning based point forecasting methods:

- DeepGLO [48] is a hybrid model which combines global matrix factorization, regularized using temporal convolution with another temporal network that can

model local properties of each individual time-series.

- N-BEATS [49] (Neural Basis Expansion Analysis for Time-Series) is an interpretable, univariate model, which is built using backward and forward residual connections and deep stack of fully-connected layers.
- FC-GAGA [278] (Fully Connected GAted Graph Architecture) is composed of fully connected hard graph gating combined with N-BEATS.

Deep learning based probabilistic forecasting methods:

- DeepAR [52] (Deep Auto-Regressive) is a RNN based probabilistic model using parametric likelihood for forecasts.
- DeepFactors [53] combines global deep learning component along with a local classical model to account for uncertainty.
- MQRNN [264] (Multi Quantile Recurrent Neural Network) is a RNN based multiple quantile regression.

Hyperparameters and Training Setup

For our model, we use an $L = 2$ layer AGCGRU [51] as the state-transition function. The dimension of the learnable node embedding is $d_e = 10$, and the number of RNN units is $d_x = 64$. We treat ρ and σ as fixed hyperparameters and set $\rho = 1$ and $\sigma = 0$ (no process noise). We train for 100 epochs using the Adam optimizer, with a batch size of 64. The initial learning rate is set to 0.01 and we follow a decaying schedule as in [50]. Hyperparameters associated with scheduled sampling [283], gradient clipping, and early stoppng are borrowed from [50]. We set the number of particles $N_p = 1$ during training and $N_p = 10$ for validation and testing. The number of exponentially spaced discrete steps [26] for integrating the flow is $N_\lambda = 29$. For each dataset, we conduct two separate experiments minimizing the training MAE (results are used to report MAE, MAPE, RMSE, and P50QL) and the training negative log posterior probability (results are used to report CRPS, P10QL, and P90QL). In addition to the AGCGRU architecture, we also conduct experiments using DCGRU [50] and GRU [343] as alternative state transition functions. For these architectures, we keep the number of layers L and the number of RNN units d_x the same as the AGCGRU. The other hyperparameters are fixed to the same values as presented above.

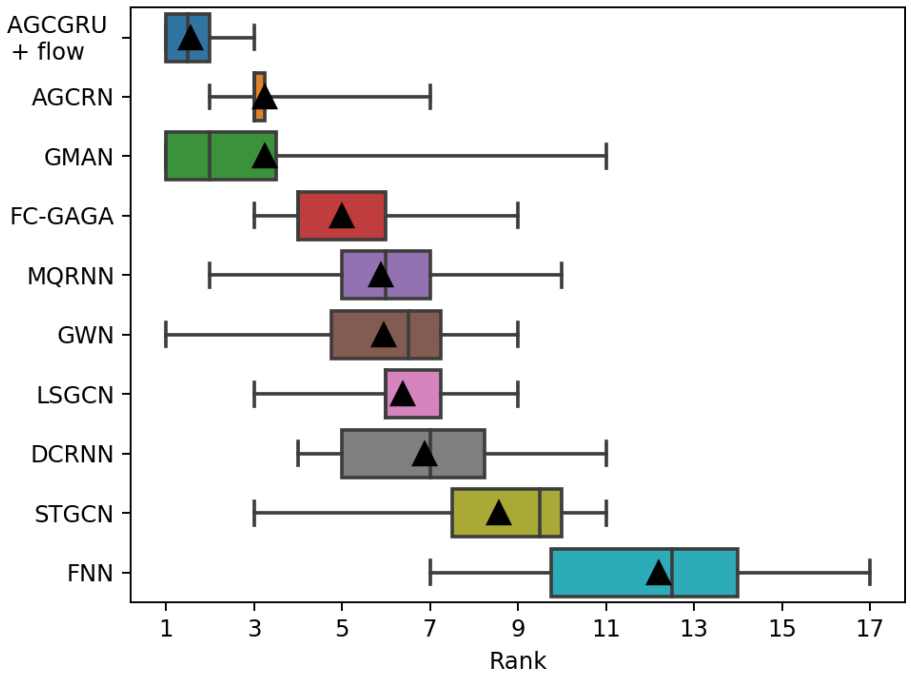
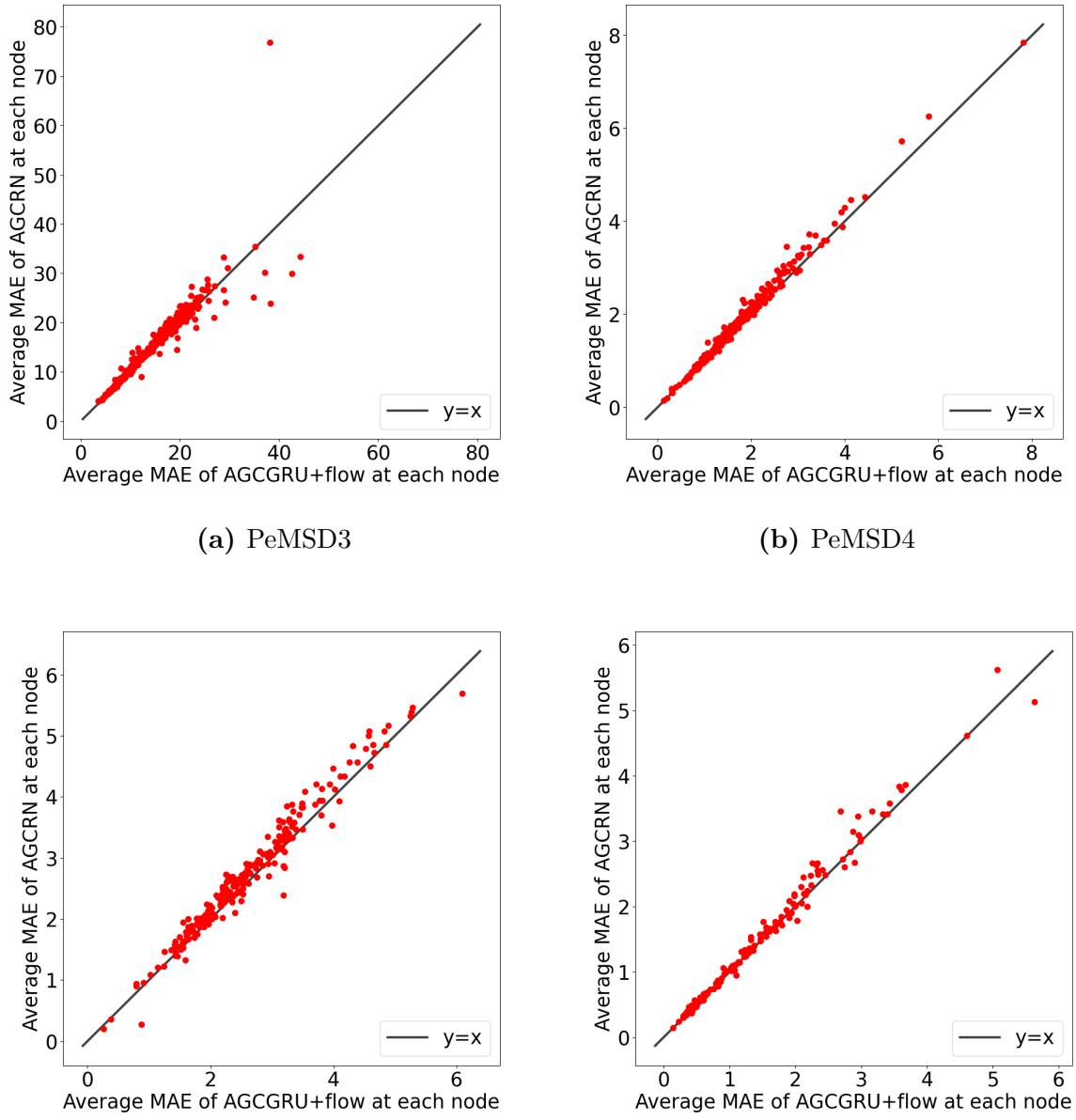


Figure 5.3: Boxplot of ranks of the top 10 algorithms across the four PeMS datasets. The means of the ranks are shown by the black triangles; whiskers extend to the minimum and maximum ranks.

Results and Discussion

The MAE, MAPE, and RMSE of predictions for 15, 30, 45, and 60 minutes horizons from all the baseline algorithms on PeMSD3 dataset are provided in Table 5.3. The results for PeMSD4, PeMSD7, and PeMSD8 are qualitatively similar and are reported in Tables C.1, C.2, and C.3 respectively in Appendix C. We present a comparison of the average rankings across datasets in Figure 5.3. We observe that most of the spatio-temporal models perform better than graph agnostic baselines in most cases. Statistical models such as HA, ARIMA, and VAR and basic machine learning models such as SVR, FNN, and FC-LSTM show poor predictive performance as they cannot model the complex spatio-temporal patterns present in the real world traffic data. Graph agnostic deep learning models such as DeepGLO and N-BEATS perform better than the statistical models, but they cannot incorporate the graph structure when learning. FC-GAGA has lower forecasting errors as it is equipped with a graph learning module. The spatio-temporal graph-based models (especially



(a) PeMSD3

(b) PeMSD4

(c) PeMSD7

(d) PeMSD8

Figure 5.4: Scatter-plots of average MAE at each node for AGCGRU+flow v.s. that of AGCRN on PeMS datasets. The AGCGRU+flow has lower average MAE compared to AGCRN at most of the nodes for all four datasets.

Table 5.3: Average MAE, MAPE and RMSE for PeMSD3 dataset for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
HA	31.58	33.78	52.39
ARIMA	17.31/22.12/27.35/32.47	16.53/20.78/25.66/30.84	26.80/34.60/42.37/49.98
VAR	18.59/20.80/23.06/24.86	19.59/21.81/24.24/26.44	31.05/33.92/36.93/39.32
SVR	16.66/20.33/24.33/28.34	16.07/19.45/23.31/27.57	25.97/32.19/38.30/44.57
FNN	16.87/20.30/23.91/27.74	19.59/23.67/30.09/35.44	25.46/30.97/36.27/41.86
FC-LSTM	19.01/19.46/19.92/20.29	19.77/20.23/20.82/21.30	32.96/33.59/34.24/34.83
DCRNN	14.42/15.87/17.10/18.29	14.57/15.78/16.87/17.95	24.33/27.05/28.99/30.76
STGCN	15.22/17.54/19.74/21.59	16.22/18.44/20.13/21.88	26.20/29.10/32.19/34.83
ASTGCN	17.03/18.50/19.58/20.95	18.02/19.28/20.18/21.61	29.04/31.81/33.98/36.37
GWN	14.63/16.56/18.34/20.08	**13.74**/15.24/16.82/18.75	25.06/28.48/31.11/33.58
GMAN	14.73/15.44/16.15/16.96	15.63/16.25/16.99/17.91	24.48/25.68/26.80/27.99
AGCRN	14.20/15.34/16.28/17.38	13.79/**14.47**/**15.14**/16.25	24.75/26.61/28.06/29.61
LSGCN	14.28/16.08/17.77/19.23	14.80/16.01/17.15/18.21	25.88/28.11/30.31/32.37
DeepGLO	14.79/18.89/19.11/23.53	14.12/16.92/17.75/21.68	22.97/29.17/30.48/35.64
N-BEATS	15.57/18.12/20.50/23.03	15.56/18.05/20.50/23.19	24.44/28.69/32.62/36.72
FC-GAGA	14.68/15.85/16.40/17.04	15.57/15.88/16.32/17.16	24.65/26.85/27.90/28.97
DeepAR	15.84/18.15/20.30/22.64	16.26/18.42/20.19/22.56	26.33/29.96/33.12/36.65
DeepFactors	17.53/20.17/22.78/24.87	19.22/24.42/29.58/34.43	27.62/31.83/35.36/37.91
MQRNN	14.60/16.55/18.34/20.12	15.17/17.34/18.94/20.66	25.35/28.77/31.50/34.40
AGCGRU+flow	**13.79**/**14.84**/**15.58**/**16.06**	14.01/14.75/15.34/**15.80**	**22.08**/**24.26**/**25.55**/**26.43**

AGCRN, GMAN, GWN, and LSGCN) display better performance. These models either use the observed graph or learn the graph structure from the data. In general, the deep learning based probabilistic forecasting algorithms such as DeepAR, DeepFactors, and MQRNN do not account for the spatial relationships in the data as well as the graph-based models, although MQRNN is among the best performing algorithms. DeepAR and DeepFactors aim to model the forecasting distributions and thus do not perform as well in the point forecasting task. The training loss function (negative log likelihood of the forecasts) does not match the evaluation metric. However, MQRNN shows better performance, possibly because it does target learning the median of the forecasting distribution along with other quantiles. The proposed AGCGRU+flow algorithm

demonstrates comparable prediction accuracy to the best-performing spatio-temporal models, such as GWN, GMAN and AGCRN and achieves the best average ranking across the four datasets. Figure 5.4 demonstrates that the proposed AGCGRU+flow has lower average MAE in most of the nodes compared to the second best performing AGCRN algorithm, for all four PeMS datasets.

Table 5.4: Average CRPS, P10QL, P50QL, and P90QL for PeMSD3 for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	CRPS (15/ 30/ 45/ 60 min)
DeepAR	11.41/13.11/14.62/16.27
DeepFactors	14.16/15.87/17.59/18.99
GRU+flow	11.23/12.70/13.98/15.25
DCGRU+flow	11.21/12.14/12.87/13.64
AGCGRU+flow	**10.53/11.39/12.03/12.47**
	P10QL(%) (15/ 30/ 45/ 60 min)
DeepAR	4.11/4.69/5.21/5.69
DeepFactors	5.85/6.33/6.91/7.51
MQRNN	4.03/4.60/5.13/5.68
GRU+flow	4.19/4.71/5.14/5.55
DCGRU+flow	4.28/4.69/4.99/5.28
AGCGRU+flow	**4.01/4.44/4.76/4.97**
	P50QL(%) (15/ 30/ 45/ 60 min)
DeepAR	9.11/10.44/11.68/13.03
DeepFactors	10.08/11.60/13.11/14.31
MQRNN	8.40/9.52/10.55/11.58
GRU+flow	8.28/9.26/10.15/11.04
DCGRU+flow	8.33/9.01/9.50/9.99
AGCGRU+flow	**7.93/8.54/8.96/9.24**
	P90QL(%) (15/ 30/ 45/ 60 min)
DeepAR	4.40/5.13/5.70/6.40
DeepFactors	6.19/6.95/7.61/8.04
MQRNN	**3.75/4.27**/4.70/5.09
GRU+flow	4.33/4.94/5.48/6.04
DCGRU+flow	4.30/4.67/4.97/5.31
AGCGRU+flow	4.06/4.38/**4.63/4.82**

Table 5.4 summarizes the results for probabilistic forecasting on PeMSD3, reporting the average CRPS, (normalized) P10QL, P50QL, and P90QL for the predictions at 15, 30, 45, and 60 minutes horizons. Qualitatively similar results for PeMSD4, PeMSD7, and PeMSD8 datasets are summarized in Tables C.4, C.5, and C.6 respectively in Appendix C. We observe that in most cases, the proposed particle flow based algorithms outperform the competitors. MQRNN also shows impressive performance in predicting the forecast quantiles, as it is explicitly trained to minimise the quantile losses. We cannot compute the CRPS for MQRNN, since instead of attempting to characterize the forecast distribution, it only provides a few discrete quantiles of the forecast. In particular, comparison of GRU+flow with the DeepAR model reveals that even without a sophisticated RNN architecture, the particle flow based approach shows better characterization of prediction uncertainty in most cases. Some qualitative visualization of the confidence intervals for 15-minute ahead predictions for the PeMSD3 dataset is shown in Figure 5.5. Similar confidence intervals for the other datasets are shown in Figures C.1, C.2, and C.3 in Appendix C. We observe that the confidence intervals from the proposed algorithm are considerably tighter compared to its competitors in most cases, whereas the coverage of the ground truth is still ensured.

Comparison to Deterministic Encoder-Decoder Models

Table 5.5 shows that in comparison to deterministic encoder-decoder based sequence to sequence prediction models, the proposed flow based approaches perform better in almost all cases for three different architectures of the RNN. In each case, both the encoder-decoder model and our approach use a 2-layer architecture with 64 RNN units. Moreover, for each RNN architecture, we use the same values for the hyperparameters associated with model training for the encoder-decoder model and the proposed particle flow based algorithm.

Comparison to the Particle Filter

In order to demonstrate the effectiveness of particle flow [20], we compare the proposed AGCGRU+flow to a Bootstrap Particle Filter (BPF) [13] based approach with the same number of particles. The use of the bootstrap particle filter leads to a computationally faster algorithm (requiring approximately 60% of the training time of the particle

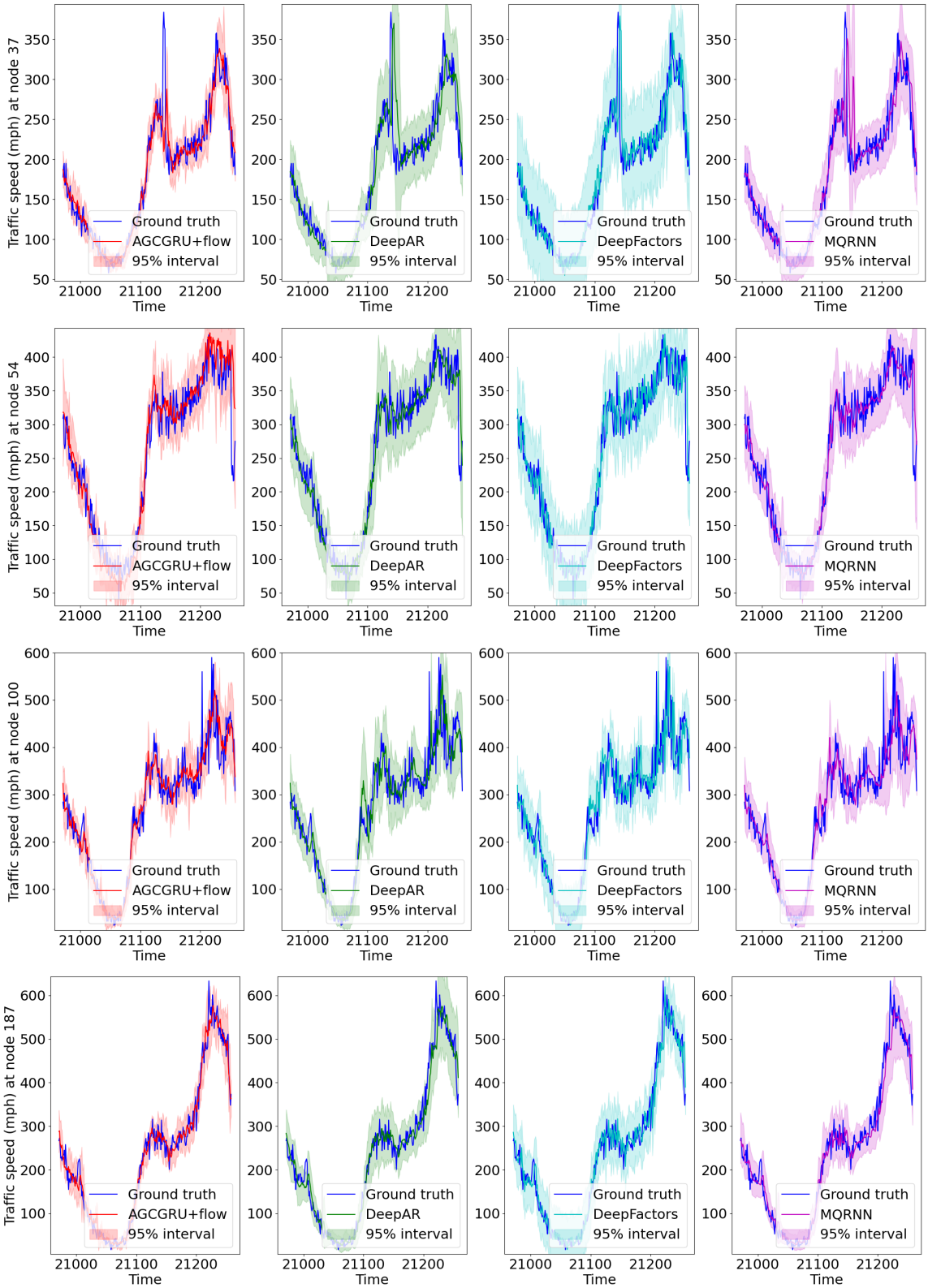


Figure 5.5: 15 minutes ahead predictions from the probabilistic forecasting algorithms with confidence intervals at nodes 37, 54, 100, and 187 of PeMSD3 dataset for the first day in the test set. The proposed AGCGRU+flow algorithm provides tighter confidence interval than its competitors in most cases, which leads to lower quantile error.

Table 5.5: Average MAE, MAPE, and RMSE for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for the proposed flow based approaches and deterministic encoder-decoder models. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**13.79**/**14.84**/**15.58**/**16.06**	**14.01**/**14.75**/**15.34**/**15.80**	**22.08**/**24.26**/**25.55**/**26.43**
FC-AGCGRU	13.96/15.37/16.52/17.45	14.26/15.61/16.69/17.37	25.28/27.43/29.09/30.43
DCGRU+flow	14.48/**15.67**/**16.52**/**17.36**	15.06/16.06/16.91/**17.84**	**23.86**/**26.12**/**27.54**/**28.76**
FC-DCGRU	**14.42**/15.87/17.10/18.29	**14.57**/**15.78**/**16.87**/17.95	24.33/27.05/28.99/30.76
GRU+flow	**14.40**/**16.10**/**17.63**/**19.18**	**14.56**/**15.99**/**17.33**/**18.89**	**23.06**/**26.15**/**28.64**/**30.97**
FC-GRU	15.82/18.37/20.61/22.93	15.87/18.82/21.32/23.75	25.85/30.09/33.37/36.94
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**1.35**/**1.63**/**1.78**/**1.88**	**2.67**/**3.44**/**3.87**/**4.16**	**2.88**/**3.77**/**4.20**/**4.46**
FC-AGCGRU	1.37/1.74/2.00/2.20	2.69/3.67/4.41/5.00	2.92/3.96/4.62/5.09
DCGRU+flow	**1.38**/**1.71**/**1.92**/**2.08**	2.72/**3.63**/**4.23**/**4.67**	**2.93**/**3.93**/**4.49**/**4.87**
FC-DCGRU	**1.38**/1.78/2.06/2.29	**2.69**/3.72/4.51/5.16	2.95/4.09/4.81/5.34
GRU+flow	**1.37**/**1.76**/**2.02**/**2.23**	**2.70**/**3.74**/**4.52**/**5.15**	**2.95**/**4.05**/**4.74**/**5.23**
FC-GRU	1.46/1.91/2.25/2.54	2.84/3.97/4.88/5.66	3.10/4.35/5.20/5.85
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**2.15**/**2.70**/**2.99**/**3.19**	**5.13**/**6.75**/**7.61**/**8.18**	**4.11**/**5.46**/**6.12**/**6.54**
FC-AGCGRU	2.21/2.99/3.56/4.05	5.18/7.39/9.12/10.64	4.18/5.88/7.03/7.94
DCGRU+flow	**2.19**/**2.87**/**3.29**/**3.61**	**5.16**/**7.17**/**8.48**/**9.42**	**4.16**/**5.66**/**6.54**/**7.14**
FC-DCGRU	2.23/3.06/3.67/4.18	5.19/7.50/9.31/10.90	4.26/6.05/7.28/8.24
GRU+flow	**2.24**/**3.02**/**3.55**/**3.96**	**5.27**/**7.58**/**9.30**/**10.60**	**4.28**/**5.97**/**7.00**/**7.73**
FC-GRU	2.41/3.40/4.17/4.84	5.60/8.27/10.47/12.40	4.56/6.68/8.17/9.34
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**1.13**/**1.37**/**1.49**/**1.57**	**2.30**/**3.01**/**3.40**/**3.65**	2.59/**3.45**/**3.85**/**4.09**
FC-AGCGRU	1.16/1.48/1.70/1.87	**2.30**/3.17/3.78/4.25	**2.58**/3.53/4.12/4.54
DCGRU+flow	1.17/**1.44**/**1.58**/**1.70**	2.35/**3.12**/**3.57**/**3.87**	2.64/3.54/**4.00**/**4.28**
FC-DCGRU	**1.16**/1.49/1.70/1.87	**2.25**/3.16/3.85/4.37	**2.54**/**3.49**/4.08/4.49
GRU+flow	**1.12**/**1.41**/**1.59**/**1.74**	**2.17**/**2.94**/**3.50**/**3.92**	**2.55**/**3.47**/**4.02**/**4.40**
FC-GRU	1.20/1.56/1.81/2.02	2.29/3.09/3.70/4.22	2.63/3.61/4.24/4.73

flow-based method). Tables 5.6 and 5.7 provide the detailed comparison both in terms of point forecasting and probabilistic forecasting metrics. We observe that the proposed AGCGRU+flow algorithm outperforms the particle filter based approach in most cases.

Table 5.6: Average MAE, MAPE, and RMSE for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow and AGCGRU+BPF. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**13.79/14.84/15.58/16.06**	**14.01/14.75/15.34/15.80**	**22.08/24.26/25.55/26.43**
AGCGRU+BPF	14.19/15.13/15.85/16.35	14.21/14.86/15.40/15.82	25.69/27.38/28.51/29.26
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**1.35/1.63/1.78/1.88**	**2.67/3.44/3.87/4.16**	**2.88/3.77/4.20/4.46**
AGCGRU+BPF	1.36/1.65/1.80/1.90	2.71/3.46/3.90/4.18	2.91/3.81/4.25/4.52
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**2.15**/**2.70**/2.99/3.19	**5.13/6.75/7.61/8.18**	**4.11/5.46/6.12/6.54**
AGCGRU+BPF	2.19/2.73/2.99/**3.17**	5.27/6.86/7.69/8.21	4.18/5.52/6.16/6.53
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**1.13/1.37/1.49/1.57**	**2.30/3.01/3.40/3.65**	**2.59/3.45/3.85/4.09**
AGCGRU+BPF	1.18/1.41/1.52/1.59	2.47/3.13/3.50/3.74	2.69/3.53/3.92/4.15

Comparison to Ensembles

We compare the proposed approach with an ensemble of competitive deterministic forecasting techniques for both point forecasting and probabilistic forecasting tasks. We choose the size of the ensemble so that the algorithms have an approximately equal execution time. We use AGCRN and GMAN to form the ensembles, as they are the best point-forecast baseline algorithms. From Table 5.8, we observe that our approach is comparable or slightly worse compared to the ensembles in terms of the MAE, MAPE and RMSE of the point forecasts. However, the proposed AGCGRU+flow shows better

Table 5.7: Average CRPS, P10QL, and P90QL for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow and AGCGRU+BPF. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	**10.53**/**11.39**/**12.03**/**12.47**	**4.01**/**4.44**/**4.76**/**4.97**	**4.06**/**4.38**/**4.63**/**4.82**
AGCGRU+BPF	11.32/11.94/12.55/12.92	4.36/4.66/4.98/5.13	4.39/4.65/4.88/5.07
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	**1.08**/1.32/1.46/1.56	**1.28**/1.62/1.82/1.97	**1.05**/1.26/1.37/1.45
AGCGRU+BPF	1.10/1.32/**1.45**/**1.54**	1.29/**1.60**/**1.79**/**1.92**	1.06/1.26/1.37/1.45
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	**1.73**/**2.18**/**2.43**/**2.58**	**2.27**/**2.97**/**3.36**/**3.60**	**1.83**/**2.25**/**2.48**/**2.62**
AGCGRU+BPF	1.79/2.24/2.49/2.66	2.35/3.02/3.40/3.67	1.86/2.29/2.53/2.69
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	**0.90**/**1.10**/**1.20**/1.28	**1.10**/**1.43**/**1.61**/**1.73**	**0.87**/**1.01**/1.09/1.14
AGCGRU+BPF	0.96/1.13/1.22/1.28	1.19/1.47/1.63/1.74	0.91/1.03/1.09/**1.13**

characterization of the prediction uncertainty compared to the ensemble methods in almost all cases, as shown in Table 5.9.

5.4.4 Experiments on Non-Graph Datasets

Point Forecasting Results on Non-Graph Datasets

We evaluate our proposed flow-based RNN on the Electricity and Traffic datasets, following the setting described in Appendix C.4 in [49]. We augment the results table in [49] with the results from an FC-GRU (a fully connected GRU encoder-decoder) and GRU+flow. We use a 2 layer GRU with 64 RNN units in both cases. We follow the preprocessing steps described by Oreshkin et al. in [49]. In the literature, four different data splits have been used for the Electricity dataset, and three different splits have been used for the Traffic dataset. Since the official implementations of the baseline methods such as DeepAR, DeepState, and DeepFactors are not publicly available and different baselines

Table 5.8: Average MAE, MAPE, and RMSE for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCRN-ensemble, GMAN-ensemble, and AGCGRU+flow. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCRN-ensemble	14.21/15.12/15.73/16.22	**13.91**/**14.56**/**14.93**/**15.38**	25.49/27.16/28.20/28.90
GMAN-ensemble	14.48/15.20/15.90/16.66	15.01/15.64/16.41/17.36	23.96/25.20/26.31/27.44
AGCGRU+flow	**13.79**/**14.84**/**15.58**/**16.06**	14.01/14.75/15.34/15.80	**22.08**/**24.26**/**25.55**/**26.43**
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCRN-ensemble	1.35/1.61/1.76/1.91	2.75/3.40/3.79/4.17	2.89/3.65/4.09/4.47
GMAN-ensemble	**1.33**/**1.57**/**1.72**/**1.84**	**2.64**/**3.27**/**3.70**/**4.04**	2.89/**3.62**/**4.04**/**4.33**
AGCGRU+flow	1.35/1.63/1.78/1.88	2.67/3.44/3.87/4.16	**2.88**/3.77/4.20/4.46
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCRN-ensemble	2.17/**2.69**/**2.95**/3.20	5.25/**6.75**/**7.55**/8.22	**4.09**/**5.29**/**5.94**/**6.45**
GMAN-ensemble	2.42/2.80/3.08/3.35	6.08/7.18/8.00/8.74	4.68/5.54/6.08/6.51
AGCGRU+flow	**2.15**/2.70/2.99/**3.19**	**5.13**/**6.75**/7.61/**8.18**	4.11/5.46/6.12/6.54
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCRN-ensemble	1.19/1.36/1.46/1.58	2.67/3.10/3.38/3.68	2.88/3.41/3.76/4.06
GMAN-ensemble	**1.13**/**1.28**/**1.39**/**1.49**	2.37/**2.78**/**3.10**/**3.37**	2.71/**3.25**/**3.61**/**3.87**
AGCGRU+flow	**1.13**/1.37/1.49/1.57	**2.30**/3.01/3.40/3.65	**2.59**/3.45/3.85/4.09

might have used different preprocessing steps to report the results in the corresponding papers, Oreshkin et al. compiled results for different splits from the original papers introducing the baseline algorithms. If the result for an algorithm using a particular split does not appear in any of those papers, 'n/a' (not available) is used to denote that scenario. We follow this approach and run the GRU+flow and FC-GRU algorithms for all splits. The evaluation metric is P50QL (Normalized Deviation) in this experiment.

From Table 5.10, we observe that the flow based approach performs comparably or better than the state-of-the-art N-BEATS algorithm for the Electricity dataset, even with a simple GRU as the state transition function. The better performance of the univariate N-BEATS compared to the multivariate models suggests that most time-series in these

Table 5.9: Average CRPS, P10QL, and P90QL for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCRN-ensemble, GMAN-ensemble, and AGCGRU+flow. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCRN-ensemble	12.64/13.44/13.96/14.27	6.90/7.40/7.54/7.53	6.10/6.43/6.79/6.96
GMAN-ensemble	12.79/13.49/14.13/14.77	7.17/7.67/8.08/8.45	5.86/6.16/6.44/6.68
AGCGRU+flow	**10.53/11.39/12.03/12.47**	**4.01/4.44/4.76/4.97**	**4.06/4.38/4.63/4.82**
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCRN-ensemble	1.20/1.44/1.56/1.68	1.82/2.21/2.39/2.57	1.53/1.82/1.93/2.08
GMAN-ensemble	1.16/1.38/1.51/1.62	1.73/2.11/2.35/2.54	1.45/1.70/1.82/1.92
AGCGRU+flow	**1.08/1.32/1.46/1.56**	**1.28/1.62/1.82/1.97**	**1.05/1.26/1.37/1.45**
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCRN-ensemble	1.90/2.39/2.60/2.81	3.22/4.15/4.55/4.89	2.55/3.19/3.35/3.58
GMAN-ensemble	1.96/2.31/2.53/2.73	3.16/3.83/4.23/4.53	2.20/2.59/2.81/3.00
AGCGRU+flow	**1.73/2.18/2.43/2.58**	**2.27/2.97/3.36/3.60**	**1.83/2.25/2.48/2.62**
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCRN-ensemble	1.03/1.20/1.28/1.38	1.63/1.97/2.14/2.34	1.18/1.34/1.39/1.48
GMAN-ensemble	0.95/**1.10/1.19/1.28**	1.40/1.68/1.88/2.04	1.12/1.26/1.34/1.41
AGCGRU+flow	**0.90/1.10**/1.20/**1.28**	**1.10/1.43/1.61/1.73**	**0.87/1.01/1.09/1.14**

datasets do not provide valuable additional information for predicting other datasets. This is in contrast to the graph-based datasets, where the performance of N-BEATS was considerably worse than the multivariate algorithms. The proposed flow-based algorithm achieves prediction performance on the Traffic dataset that is comparable to N-BEATS except for one split with limited training data. Across all datasets and split settings, our flow-based approach significantly outperforms the FC-GRU. The proposed algorithm outperforms TRMF, DeepAR, DeepState and DeepGLO. It outperforms DeepFactors for the Electricity dataset, but is worse for the Traffic dataset (for the same split with limited available training data).

Table 5.10: Normalized Deviation on Electricity and Traffic datasets. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	Electricity				Traffic		
	2014-09-01	2014-03-31	2014-12-18	last 7 days	2008-06-15	2008-01-14	last 7 days
TRMF	0.160	n/a	0.104	0.255	0.200	n/a	0.187
DeepAR	0.070	0.272	0.086	n/a	0.170	0.296	0.140
DeepState	0.083	n/a	n/a	n/a	0.167	n/a	n/a
DeepFactors	n/a	0.112	n/a	n/a	n/a	**0.225**	n/a
DeepGLO	n/a	n/a	<u>0.082</u>	n/a	n/a	n/a	0.148
N-BEATS	**0.064**	**0.065**	n/a	<u>0.171</u>	**0.114**	<u>0.230</u>	**0.110**
FC-GRU	0.102	0.118	0.098	0.203	0.259	0.528	0.233
GRU+flow	<u>0.070</u>	<u>0.071</u>	**0.069**	**0.140**	<u>0.133</u>	0.322	<u>0.125</u>

Probabilistic Forecasting Results on Non-Graph Datasets

For comparison with state-of-the-art deep learning based probabilistic forecasting methods on standard non-graph time-series datasets, we evaluate the proposed GRU+flow algorithm following the setting in [269]. The results reported in Table 1 of [269] are augmented with the results of the GRU+flow algorithm. We use a 2 layer GRU with 64 RNN units in each case. We follow the preprocessing steps as in [265, 269]. The evaluation metric is (normalized) CRPS_{sum}, which is obtained by first summing across the different time-series, both for the ground-truth test data, and samples of forecasts, and then computing the (normalized) CRPS on the summed data. The results are summarized in Table 5.11. We observe that the proposed GRU+flow achieves the lowest CRPS_{sum} for all datasets.

Table 5.11: Average CRPS_{sum} for Electricity, Traffic, Taxi, and Wikipedia datasets. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better

Dataset	Vec-LSTM ind-scaling	Vec-LSTM lowrank-Copula	GP scaling	GP Copula	LSTM Real-NVP	LSTM MAF	Transformer MAE	GRU+ flow
Electricity	0.025	0.064	0.022	0.024	0.024	<u>0.021</u>	<u>0.021</u>	**0.013**
Traffic	0.087	0.103	0.079	0.078	0.078	0.069	<u>0.056</u>	**0.028**
Taxi	0.506	0.326	0.183	0.208	0.175	<u>0.161</u>	0.179	**0.140**
Wikipedia	0.133	0.241	1.483	0.086	0.078	0.067	<u>0.063</u>	**0.054**

5.4.5 Computational Complexity, Memory Requirement, and Execution Time

For simplicity, we consider a GRU instead of a graph convolution based RNN and we only focus on one sequence instead of a batch. Our model has to perform both GRU computation and particle flow for the first P time steps and then apply the GRU and the linear projection for the next Q steps to generate the predictions. For an L-layer GRU with d_x RNN units and N-dimensional input, the complexity of the GRU operation for N_p particles is $\mathcal{O}((P+Q)N_pLNd_x^2)$ [343]. The total complexity of the EDH particle flow [103] is $\mathcal{O}(PN_\lambda N^3)$ for computing the flow parameters and $\mathcal{O}(PN_pN_\lambda Nd_x^2)$ for applying the particle flow. The total complexity of the measurement model for N_p particles is $\mathcal{O}(QN_pNd_x^2)$. Since in most cases $N \gg d_x$ and $N \gg N_p$, the complexity of our algorithm for forecasting of one sequence is $\mathcal{O}(PN_\lambda N^3)$. Many of the other algorithms exhibit a similar $\mathcal{O}(N^3)$ complexity, e.g. TRMF, AGCRN, GMAN.

Table 5.12: Execution time, memory consumption (during training) and model size for AGCRN-ensemble, GMAN-ensemble, and AGCGRU+flow for the four PeMS datasets. Lower numbers are better.

Algorithm	Execution time (minutes)			
	PEMS03	PEMS04	PEMS07	PEMS08
AGCRN-ensemble	369	243	183	224
GMAN-ensemble	444	224	195	185
AGCGRU+flow	**325**	**205**	**154**	**177**
Algorithm	GPU memory (GB)			
	PEMS03	PEMS04	PEMS07	PEMS08
AGCRN-ensemble	**6.55**	**5.19**	**4.09**	**3.47**
GMAN-ensemble	15.45	9.45	8.46	4.45
AGCGRU+flow	25.27	18.76	12.45	8.45
Algorithm	Model Size (MB)			
	PEMS03	PEMS04	PEMS07	PEMS08
AGCRN-ensemble	11.52	11.52	11.45	11.45
GMAN-ensemble	**9.54**	**9.51**	**9.45**	**9.35**
AGCGRU+flow	12.88	12.86	12.86	12.85

Table 5.12 reports the run time, GPU usage during training, and the size of the learned model for AGCRN-ensemble, GMAN-ensemble, and the proposed AGCGRU+flow for the four PeMS datasets. We observe that if we choose the ensemble size so that the algorithms have an approximately equal execution time, then the model-size of the ensemble algorithms are comparable to our approach as well. However, our method requires more GPU memory compared to the ensembles during training because of the particle flow in the forward pass.

5.5 Summary

In this chapter, we propose a state-space probabilistic modeling framework for multivariate time-series prediction that can process information provided in the form of a graph that specifies (probable) predictive or causal relationships. We develop a probabilistic forecasting algorithm based on the Bayesian inference of hidden states via particle flow. For spatio-temporal forecasting, we use GNN-based RNN architectures to instantiate the framework. Our method demonstrates comparable or better performance in point forecasting and considerably better performance in uncertainty characterization compared to existing techniques.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

The overarching goal of this thesis is to derive scalable, effective, and computationally efficient Monte Carlo methods for carrying out approximate Bayesian inference in diverse tasks. We have considered three different problem settings: a) conducting sequential inference using particle flow in the presence of noise distributed as Gaussian mixtures, b) improving graph-based learning tasks by proposing Bayesian approaches to account for the uncertainty associated with the graph structure, and c) designing novel probabilistic spatio-temporal forecasting algorithms using a combination of graph-aware recurrent architectures and particle flow.

Chapter 2 provides the necessary background material and detailed literature review related to these research areas. The main novel contributions of this thesis are presented in Chapters 3, 4, and 5. We summarize the key aspects of the proposed methodology and obtained results in the following sections.

6.1.1 Sequential Inference in Presence of Gaussian Mixture Noise Models

In Chapter 3, we investigate effective nonlinear filtering techniques for multi-modal posterior distributions, which arises due to the Gaussian mixture densities of the dynamic and/or measurement noises [30,31] in a state-space model. Unimodal approximation of the posterior in such cases leads to inaccurate state estimates [29]. We use particle flow [21, 105] as a key ingredient in the design of the proposed algorithms, since it has a demonstrably superior capability of representing high-dimensional and/or highly localized posterior distributions compared to many existing approaches. We propose three novel algorithms; the PF-GMM, the PFPF-GMM, and the SmHMC-GMM (LEDH).

The PF-GMM approach approximates the state posterior by a Gaussian mixture distribution at each time step where each component mean is computed using a separate LEDH [105] particle flow and the mixture proportions and the covariance matrices are obtained from

the extended Kalman filter implementation of the Gaussian sum filter [30]. Resampling of GMM components is employed to prevent the exponential growth of the number of mixture components over time. This approach scales in the same manner as LEDH with the state dimension, whereas existing particle flow algorithms for mixture models [31] struggle in high dimensions.

Although the PF-GMM achieves impressive empirical performance, particle flow cannot yield true samples from the posterior distribution due to the potentially mismatched modelling assumptions used for the derivation of the flow equation and the numerical approximations employed for solving it. In order to obtain a statistically consistent alternative solution, we adapt the *particle flow particle filter* (PFPF) [26] to our problem setting by reinterpreting the hidden Markov model with Gaussian mixture distributed noises as a switching state-space model. The PFPF algorithm conducts an auxiliary particle flow with a deterministic, invertible mapping property and use it to construct a proposal distribution for a particle filter. The resulting proposal is easy to evaluate because of the invertible mapping established by the flow and is expected to be well-matched to the target posterior due to the effectiveness of particle flow methods in transporting the particles to the high posterior density regions of the state-space. The proposed PFPF-GMM algorithm augments the switching variables to the state vector to perform importance sampling in an extended state-space using a invertible particle flow based conditional proposal distribution for the state.

As another extension, we derive a *sequential MCMC* (SMCMC) [15, 344] method, which exhibits superior performance compared to the importance sampling based PFPF-GMM algorithm in high dimensions. The proposed SmHMC-GMM (LEDH) algorithm combines efficient, measurement-driven proposal distributions for the discrete switching variables with the invertible particle flow based proposal for the state to perform the joint draw step inside the composite kernel based SMCMC framework. This leads to efficient traversal of different modes of the state posterior. The subsequent *Metropolis within Gibbs* refinement step for the state uses a recent MCMC technique, termed *manifold Hamiltonian Monte Carlo* (mHMC) [129], to further improve the quality of the posterior approximation by local exploration of the state-space. We also present some theoretical convergence results for the SMCMC methods in Appendix A.

Empirical results reveal that the proposed algorithms closely match the performance of the

(almost) optimal performance in the linear setting and achieve considerably more accurate state-estimates for the nonlinear state-space model, whereas many existing particle flow and particle filters fail abysmally, either due to the use of implicit Gaussian or unimodal assumption for the target posterior in their derivation, or because of the weight degeneracy in high dimensions.

6.1.2 Bayesian Graph Convolutional Neural Networks

In Chapter 4, we consider learning tasks on graph structured data, where some additional information, encoded by a graph that represents data points as nodes and the relationships as edges, is available for model training. However, the provided graph is likely to be incomplete and/or contain spurious edges due to a noisy measurement process or the lack of knowledge concerning its generative mechanism. In such cases, incorporating the uncertainty associated with graph structure into learning can result in improved performance. To that end, we strive to achieve scalable and effective Bayesian modelling and inference for the graph topology. We propose a Bayesian Graph Convolutional Networks (BGCN) framework which views the observed graph as a realization from a suitable parametric family of random graphs, targets inference of the 'true' graph connectivity, and combines this procedure with the GCN [37]. For a node classification problem, we instantiate this approach by modelling the observed graph using an *assortative mixed membership stochastic block model* (a-MMSBM) [200, 326], which strives to capture the community structure in homophilic graphs. The resulting BGCN (MMSBM) algorithm provides better performance when there are very few node labels available during the training process. It is also shown to have improved resilience to adversarial node attacks, compared to its non-Bayesian version. To the best of our knowledge, this was the first attempt to propose incorporation of the Bayesian inference of graphs in graph neural networks.

Despite its effectiveness, parametric modelling of graph topology has several disadvantages. Choice of a suitable random graph model is heavily task-dependent and often the parameter inference of the chosen model is computationally expensive for large graphs. Another severe shortcoming of our proposed approach is that if the parametric graph model is not dependent on training data, such as node/edge features and/or labels, then

the graph inference procedure disregards them. Although subsequent works address some of these issues, they either still use parametric graph models [41] or form a variational approximation [43] of the graph posterior. Moreover, these approaches focus solely on the node classification task.

By contrast, we propose to incorporate a non-parametric graph inference step into the graph-based learning tasks. In this non-parametric graph model, a higher edge weight is more likely if the incident nodes are more similar to each other. The notion of similarity between nodes depends on the nature of the node/edge level learning objective and is measured in terms of a suitably designed distance between them. This non-parametric graph learning approach scales gracefully to larger graphs and the flexibility in designing the pairwise distance matrix among nodes allows its use in other graph related tasks in addition to node classification. Using this approach, we observe increased accuracy of node classification for settings where the amount of labeled data is very limited. For the setting of unsupervised learning, we demonstrate that incorporating a graph learning step when performing variational modelling of the graph structure with autoencoder models leads to better link prediction.

6.1.3 RNN with Particle Flow

The methodology presented in Chapter 5 addresses the task of spatio-temporal forecasting, where topological information about the relationships among different time-series are available or learned from the data and is effectively utilized to achieve superior performance. With contemporaneous research efforts on developing sophisticated neural networks suitable for graph structured data, recent years have seen a tremendous amount of focus on deriving spatio-temporal forecasting [50, 51, 276, 278] algorithms. Despite the empirical success in providing accurate point forecasts, a major limitation of these algorithms is that we cannot construct confidence intervals around the obtained predictions. In order to remedy this issue, we propose a novel probabilistic forecasting framework. The time-series is assumed to be a random observation sequence from a graph cognizant, nonlinear state-space model, whose parameters are to be learned from the training data. The state-transition in our model is represented by recent graph convolutional recurrent architectures [50, 51], which show commendable accuracy in point forecasting. The randomness present in the state-transition and/or emission mechanisms

within the state-space model allows us to incorporate the uncertainty in the forecasts seamlessly in our method. Forecasting using our model is equivalent to computing the posterior predictive distribution of the forecasts. This requires computation of an integral, which is an expectation with respect to the posterior distribution of the hidden state of the model. In order to deal with the intractability of this integral, we resort to Monte Carlo sampling. Motivated by its impressive performance in high-dimensional nonlinear filtering, we employ particle flow [21] as the tool for posterior inference of the hidden states. During model training, we aim to minimise a point forecasting criterion, such as mean absolute error, if the model is to be evaluated using a performance metric based on such a criterion. For probabilistic forecasting, we derive a Monte Carlo approximation of the negative log-likelihood of the model parameters, which is used as a loss function in an end to end learning setup. The resulting algorithm combines graph convolutions, recurrent architectures, and particle flow.

The general framework we propose is extremely flexible. It can be adapted to the univariate or multivariate setting, can handle additional covariates, and has the capability of processing an observed graph or learning one from the data depending on the choice of the deployed recurrent architecture. Moreover, sophisticated nonlinear filtering algorithms can be used for inference of the state-posterior.

We conduct extensive point and probabilistic forecasting experiments on several real-world, graph-based and non-graph, multivariate time-series datasets. For the point forecasting task for the PeMS traffic datasets, a specific instantiation of the proposed approach achieves the lowest average rank, outperforming a wide range of baseline algorithms. In comparison to existing probabilistic forecasting techniques, the proposed flow based RNN models obtain considerably lower CRPS and quantile errors. Our algorithms attain lower errors compared to the corresponding deterministic encoder-decoder models in most cases. Comparison with ensembles of state-of-the-art spatio-temporal point forecasting models reveals the superiority of the proposed algorithms in characterizing the prediction uncertainty. The choice of using particle flow instead of using a particle filter is shown to be justified empirically. For the datasets with no graphs, a GRU-based implementation of our framework performs comparably to the state-of-the-art NBEATS [49] algorithm for point forecasting, whereas it yields substantially lower CRPS_{sum} in comparison to a state-of-the-art normalizing flow-based probabilistic forecasting algorithm [269].

6.2 Future Work

The research presented in this thesis can be improved and extended in several ways. We list some potential future research directions in the following sections.

6.2.1 Sequential Inference in Presence of Gaussian Mixture Noise Models

In Chapter 3, the numerical simulations are conducted in an ideal setting that ignores important practical considerations. We have chosen to report the results for the ideal scenario because it allows us to focus on the sampling capabilities of the algorithms and present a fair comparison with state-of-the-art baseline techniques. The incorporation of such deviations from the idealized assumptions into the simulations is definitely important, but in a first performance assessment it can obfuscate or exaggerate the advantage of the proposed algorithms, as these imperfections typically affect the performance of different algorithms in a non-uniform way.

One such example of the real-world challenges in sensor networks is data incest due to the inadvertent re-use of the same measurements [345], which can be mitigated to some extent by a data incest management strategy that takes into account the network topology [346] or information fusion techniques with copula processes [347]. Other problems common in practical sensor networks are missed detections, false alarms, and finite measurement resolution. Addressing such practical problems is important and can lead to interesting research directions, e.g., designing an appropriate combination of sequential MCMC, particle flow and random finite sets.

Another important future research direction is a more extensive experimental evaluation of the proposed algorithms. These experiments should investigate the impact of the initial state values, the process noise variance, the measurement noise variance, coupling in the dynamic models, partial observations, the data rate and measurement uncertainty. Such experiments can shed light on the robustness of the proposed particle flow-based approaches in practical applications and motivate the development of new algorithms that address any exposed deficiencies.

Beyond experimentation, there are important methodological and theoretical issues to

explore in future work. In terms of particle flow, the PFPF-GMM and the SmHMC-GMM (LEDH) algorithms use deterministic flows in order to obtain an invertible mapping. It is more challenging to incorporate stochastic particle flow, but the stochastic flow algorithms have been demonstrated to attain considerably better performance [112], so integration is desirable. For example, an initial exploration of constructing a particle flow particle filter using a stochastic Gromov's particle flow is considered in [348], which leads to increased diversity of the particles. This approach could be adapted to modify the PFPF-GMM and the SmHMC-GMM (LEDH) algorithms. Theoretical and empirical results suggest that non-reversible Markov processes have better mixing properties and thus provide estimates of ergodic averages with lower variance [349]. Use of recently proposed non-reversible MCMC algorithms [350–353] for the individual refinement steps of SMCMC, as shown in [354], could prove to be beneficial.

The stiffness of the differential equations is an issue in particle flow as it can lead to numerical instability. For this reason, and also to guarantee the invertible mapping property for the flows, we use a very small step size in the particle flow procedure. This leads to a greater computational overhead. Alternative strategies for addressing stiffness have been proposed in [108, 112], and it would be interesting to explore their incorporation in the flow-based SMCMC framework. We focus on the asymptotic convergence results of the proposed algorithms. Finite sample analysis of filter errors is an important direction to explore; a valuable finite sample bound for SMCMC errors is provided in [344]. In a similar manner to [355], it may be possible to identify a relationship between the magnitude of the error and the stiffness of the flow.

6.2.2 Bayesian Graph Convolutional Neural Networks

The methodology presented in Chapter 4 is primarily suited for learning in a transductive setting, i.e., it requires that all nodes in the graph are present during model training. For example, in the BGCN (MMSBM) algorithm, the features and connectivity of the nodes in the test set are used in the inference of MMSBM parameters and GCN-based feature aggregation. The non-parametric approach is no exception, since we need to compute the pairwise distances between all nodes for estimating the MAP graph. Other work [41, 43] that considers graph inference for improving graph-based learning also operates in the

transductive setup. However, inductive learning is required in many cases. In many settings, nodes are added to and/or removed from the graph. Generalization across graphs with the same form of features is required. For extremely large graphs, adopting a transductive setup is infeasible due to the high computational burden and storage requirements. Development of Bayesian approaches for inductive graph neural networks [136] is thus an important direction for future research.

Both the assortative MMSBM and the non-parametric graph model rely on the homophily of the underlying 'true' graph. SBM approaches strive to learn the community structure imposed by the node labels in a homophilic graph, whereas the non-parametric model assigns higher edge-weight between two nodes if a suitable distance between them is lower. Using these approaches in conjunction with local neighborhood aggregation based GNN models, such as the GCN is thus advantageous for node classification in homophilic datasets. However, the proposed BGCN implementations are not suitable for disassortative graphs, where the relationships among nodes are more complex and capturing meaningful long-range dependencies are crucial for the learning task. Choosing suitable statistical models for such graphs, designing efficient inference procedures, and combining the graph inference with structure aware GNNs, such as the Geom-GCN [356], are some promising avenues for future explorations.

Our work deals with Bayesian modelling of a static graph. However, in many application domains, such as social networks, financial transactions, and recommender systems, the graph is inherently time-varying. Effective GNN architectures to process the time-varying graphs are investigated in [357–359]. Extending our approach to such GNN models via Bayesian modelling of time-evolving graph structures can be undertaken in future work.

In recent years, various versions of our Bayesian approach have been adopted in diverse problem settings, such as recommender systems [57, 211, 360], multi-instance learning [361], and spatio-temporal forecasting [362]. Identifying such novel application areas, where the inference of the graph topology can improve performance, is of great practical interest and can lead to many interesting applications of our methodology in the future.

6.2.3 RNN with Particle Flow

For the numerical experiments in Chapter 5, the largest dataset we have considered is Wikipedia, which is a multivariate time-series with 2000 dimensions. Scaling the proposed methodology to extremely high dimensional settings is of significant importance and can be addressed in several ways. For spatio-temporal predictions using the graph-based recurrent architectures, this can be achieved if the graph can be partitioned meaningfully. For non-graph datasets, we can use the cross-correlation among different time-series to group them into several lower-dimensional problems. Alternatively, we can train a univariate model based on all the time-series as in [53, 54], at the expense of losing the capability to model any potential relationships among the time-series.

Although the particle flow based inference of the hidden states exhibits impressive performance in our experimental studies, our approach cannot generate true samples from the joint distribution of the forecasts even if the time-series are observations from a state-space model, whose parameters are known and available while computing the integral in eq. (5.7). This is due to the approximations employed in the particle flow implementation, as discussed in Section 2.1.6. Incorporating sophisticated particle filters into our framework is thus desirable. For example, we could use the particle flow particle filter [26] algorithms, which enjoy the impressive performance of the particle flow, while ensuring statistical consistency like any other particle filter. However, in most practical particle filters, there is a resampling step, which results in non-differentiability of the training loss function with respect to model parameters in learning problems involving particle filters. Although several existing particle filter based learning approaches [225–227] ignore this dependence of the resampling probabilities on model parameters, recent advances in differentiable particle filtering [363, 364] provide principled ways to address this issue. Incorporation of these ideas in our framework could lead to improved forecasting.

Our probabilistic approach can incorporate any RNN since the time-evolution in recurrent architectures can be interpreted as a state-transition in a state-space model. However, many *state-of-the-art* point forecasting techniques use different architectures, e.g., temporal convolution [276, 286], attention mechanisms [293], and residual connections [49, 278]. Development of probabilistic frameworks for such models is another promising direction for

future research endeavors.

Appendix A
Convergence Results for SMCMC

In this chapter, we provide theoretical results regarding asymptotic convergence of the SMCMC algorithms. These results are not restricted to the invertible particle flow composite kernel case, but hold for SMCMC methods provided the kernel and filtering problem satisfy certain assumptions (see Section A.1). While similar in spirit to results presented in [127] for the SIMCMC algorithm, the key assumptions are slightly less restrictive and the theorem directly addresses the sequential implementation in [15, 55, 61]. Recently, Finke et al. carry out a rigorous statistical analysis of SMCMC algorithms to establish upper bounds on finite sample filter errors, in addition to providing asymptotic convergence results and a central limit theorem for the SMCMC based estimators in [344].

A.1 Introduction

We present some theoretical results regarding convergence of the approximate joint posterior distribution $\widehat{\pi}_k(x_{0:k})$ obtained from SMCMC to the true joint posterior $\pi_k(x_{0:k})$ for every $k \geqslant 0$. As in Section 2.1.8, we use $\pi_k(x_{0:k})$ to denote our target distribution $p(x_{0:k}|z_{1:k})$. We initialize by setting $\pi_0(x_0) = p(x_0)$. To facilitate concise presentation of the results, we use a simplified notation in this chapter, where $x^i_{0:k}$ and $x^{*(i)}_{0:k}$ denote $x^i_{k,0:k}$ and $x^{*(i)}_{k,0:k}$, respectively. We also use $\widehat{\pi}^{(N_p)}_k$ and $\breve{\pi}^{(N_p)}_k$ to denote $\widehat{\pi}_k$ and $\breve{\pi}_k$ to indicate explicitly that they are comprised of N_p MCMC samples. While proving the theorems, we assume that the burn-in period $N_b = 0$, for simplicity. The results are, however valid for any non-zero N_b. Our results are derived for Algorithms (2.6), (2.7), (2.9) and (2.11). However, they can be easily extended to apply to Algorithm (3.3) as well, by considering convergence in the extended state space $(x_{0:k}, d_{1:k}, c_{1:k})$ which implies convergence of $x_{0:k}$.

We assume that $\pi_k(x_{0:k})$ is defined on a measurable space $(E_k, \mathcal{F}_k)$ where $E_0 = E$, $\mathcal{F}_0 = \mathcal{F}$, and $E_k = E_{k-1} \times E$, $\mathcal{F}_k = \mathcal{F}_{k-1} \times \mathcal{F}$. We denote by $\mathcal{P}(E_k)$ the set of probability measures on $(E_k, \mathcal{F}_k)$. We also define $\mathcal{S}_k = \{x_{0:k} \in E_k : \pi_k(x_{0:k}) > 0\}$. For $k \geqslant 0$, π_k is known up to a

normalizing constant $0 < Z_k < \infty$. We have

$$\pi_k(x_{0:k}) = \frac{p(x_0)\prod_{l=1}^{k} p(x_l|x_{l-1})p(z_l|x_l)}{p(z_{1:k})} = \frac{\gamma_k(x_{0:k})}{Z_k}\,, \tag{A.1}$$

where $\gamma_k : E_k \to \mathbb{R}^+$ is known point-wise and the normalizing constant $Z_k = \int_{E_k} \gamma_k(dx_{0:k}) = p(z_{1:k})$ is the unknown marginal likelihood of the observations. For the joint draw step in SMCMC, the proposal distribution at time $k = 0$ is $q_0(x_0)$, and for $k > 0$ it is $q_k(x_{0:k-1}, x_k)$, where $q_k : E_{k-1} \times E \to \mathbb{R}^+$ is a probability density in its last argument x_k conditional on its previous arguments $x_{0:k-1}$. For $k > 0$ and for any measure $\mu_{k-1} \in \mathcal{P}(E_{k-1})$, we define

$$(\mu_{k-1} \times q_k)(dx_{0:k}) = \mu_{k-1}(dx_{0:k-1})q_k(x_{0:k-1}, dx_k)\,. \tag{A.2}$$

Based on the proposal distributions $q_0(x_0)$ and $q_k(x_{0:k-1}, x_k)$, we define importance weights:

$$w_0(x_0) = \frac{\gamma_0(x_0)}{q_0(x_0)}\,; \tag{A.3}$$

and for $k > 0$

$$w_k(x_{0:k}) = \frac{\gamma_k(x_{0:k})}{\gamma_{k-1}(x_{0:k-1})q_k(x_{0:k-1}, x_k)}\,. \tag{A.4}$$

Combining eq. (A.1), (A.2), (A.3) and (A.4), it is easy to derive that

$$Z_0 = \int_{E_0} w_0(x_0)q_0(dx_0)\,, \tag{A.5}$$

and for $k > 0$

$$\frac{Z_k}{Z_{k-1}} = \int_{E_k} w_k(x_{0:k})(\pi_{k-1} \times q_k)(dx_{0:k})\,. \tag{A.6}$$

We note that, $x_{0:k}^{*(i)}$ is distributed according to $(\widehat{\pi}_{k-1}^{(N_p)} \times q_k)(x_{0:k})$, where $\widehat{\pi}_{k-1}^{(N_p)}(x_{0:k-1})$ approximates $\pi_{k-1}(x_{0:k-1})$ and is obtained from the SMCMC algorithm in the previous

time step $k-1$. Thus, the (ratio of) normalizing constants can easily be estimated as

$$\widehat{Z_0}^{(N_p)} = \frac{1}{N_p}\sum_{i=1}^{N_p} w_0(x_0^{*(i)})\,, \tag{A.7}$$

and for $k > 0$

$$\widehat{\left(\frac{Z_k}{Z_{k-1}}\right)}^{(N_p)} = \frac{1}{N_p}\sum_{i=1}^{N_p} w_k(x_{0:k}^{*(i)})\,. \tag{A.8}$$

We use $\widehat{\pi}_k^{(N_p)}(x_{0:k})$ to denote the empirical measure approximation of the target distribution $\pi_k(x_{0:k})$.

$$\widehat{\pi}_k^{(N_p)}(x_{0:k}) = \frac{1}{N_p}\sum_{i=1}^{N_p} \delta_{x_{0:k}^i}(x_{0:k})\,. \tag{A.9}$$

For any measure μ and integrable test function $f : E \to \mathbb{R}$, we define $\mu(f) = \int_E f(x)\mu(dx)$. We denote that $\mathcal{L}^p(E_k, \mathcal{F}_k, \mu_k) = \{f_k : E_k \to \mathbb{R}$ such that f_k is measurable with respect to $\mathcal{F}_k$ and $\mu_k(|f_k|^p) < \infty\}$ for $p \geqslant 1$. In other words, $\mathcal{L}^p(E_k, \mathcal{F}_k, \mu_k)$ is the set of real-valued, $\mathcal{F}_k$-measurable functions defined on E_k, whose absolute p'th moment exists with respect to μ_k. We require the following relatively weak assumption to be satisfied:

Assumption 1. *For any time index $k \geqslant 0$, there exists $B_k < \infty$ such that for any $x_{0:k} \in \mathcal{S}_k$, we have $w_k(x_{0:k}) \leqslant B_k$.*

Our first result in Section A.2 establishes almost sure convergence of the Monte Carlo estimates to their true values. As a corollary, convergence of the normalizing constants is shown in Section A.3.

We begin with several notations and propositions in order to characterize the MCMC kernel employed in the SMCMC algorithm. For $k > 0$, the proposal distribution q_k^{opt} that minimizes the variance of importance weights is the conditional density of x_k given $x_{0:k-1}$ under π_k [127] and is defined as follows:

$$\begin{aligned} q_k^{opt}(x_{0:k-1}, x_k) = \overline{\pi}_k(x_{0:k-1}, x_k) &:= \frac{\pi_k(x_{0:k})}{\pi_k(x_{0:k-1})}\,, \\ &= p(x_k|x_{k-1}, z_k)\,, \end{aligned} \tag{A.10}$$

where $\pi_k(x_{0:k-1}) = \int_E \pi_k(x_{0:k})dx_k$. With this 'optimal' proposal density, the optimal importance weight $w_k^{opt}(x_{0:k})$ does not depend on x_k.

$$\begin{aligned} w_k^{opt}(x_{0:k}) &\propto \pi_{k/k-1}(x_{0:k-1})\,, \\ &:= \frac{\pi_k(x_{0:k-1})}{\pi_{k-1}(x_{0:k-1})} = p(z_k|x_{k-1})\,. \end{aligned} \tag{A.11}$$

In the SMCMC algorithm, at iteration i of any given time step k, there is a joint draw of $x_{0:k}^i$ which is then followed by individual refinements of $x_{0:k}^{(i)}$ using the *Metropolis within Gibbs* technique, if $k > 0$. There is no refinement step at time $k = 0$. Using eq. (2.61), (2.62), (A.3), and (A.4), we calculate the acceptance probability of the joint draw as follows:

$$\begin{aligned} \alpha_0(x_0^{i-1}, x_0^{*(i)}) &= \min\left(1, \frac{\pi_0(x_0^{*(i)})q_0(x_0^{i-1})}{q_0(x_0^{*(i)})\pi_0(x_0^{i-1})}\right)\,, \\ &= \min\left(1, \frac{w_0(x_0^{*(i)})}{w_0(x_0^{i-1})}\right)\,, \end{aligned} \tag{A.12}$$

and for $k > 0$,

$$\begin{aligned} \alpha_k(x_{0:k}^{i-1}, x_{0:k}^{*(i)}) &= \min\left(1, \frac{\breve{\pi}_k(x_{0:k}^{*(i)})\widehat{\pi}_{k-1}^{(N_p)}(x_{0:k-1}^{i-1})q_k(x_{0:k-1}^{i-1}, x_k^{i-1})}{\widehat{\pi}_{k-1}^{(N_p)}(x_{0:k-1}^{*(i)})q_k(x_{0:k-1}^{*(i)}, x_k^{*(i)})\breve{\pi}_k(x_{0:k}^{i-1})}\right)\,, \\ &= \min\left(1, \frac{\pi_k(x_{0:k}^{*(i)})\pi_{k-1}(x_{0:k-1}^{i-1})q_k(x_{0:k-1}^{i-1}, x_k^{i-1})}{\pi_{k-1}(x_{0:k-1}^{*(i)})q_k(x_{0:k-1}^{*(i)}, x_k^{*(i)})\pi_k(x_{0:k}^{i-1})}\right)\,, \\ &= \min\left(1, \frac{w_k(x_{0:k}^{*(i)})}{w_k(x_{0:k}^{i-1})}\right)\,. \end{aligned} \tag{A.13}$$

We define the independent MH kernel to initialize the algorithm, $K_0^{draw} : E_0 \times \mathcal{F}_0 \to [0,1]$ as follows:

$$K_0^{draw}(x_0, dx_0^{'}) = \alpha_0(x_0, x_0^{'})q_0(dx_0^{'}) + \left(1 - \int_{E_0} \alpha_0(x_0, y_0)q_0(dy_0)\right)\delta_{x_0}(dx_0^{'})\,. \tag{A.14}$$

For $k > 0$, we use $\widehat{\pi}_{k-1}^{(N_p)} \in \mathcal{P}_{k-1}(E_{k-1})$ to construct the joint proposal $(\widehat{\pi}_{k-1}^{(N_p)} \times q_k)$ for the joint draw, which is associated with the Markov kernel $K_k^{draw} : E_k \times \mathcal{F}_k \to [0,1]$, defined

by

$$K_k^{draw}(x_{0:k}, dx'_{0:k}) = \alpha_k(x_{0:k}, x'_{0:k})(\widehat{\pi}_{k-1}^{(N_p)} \times q_k)(dx'_{0:k}) + \left(1 - \int_{E_k} \alpha_k(x_{0:k}, y_{0:k})(\widehat{\pi}_{k-1}^{(N_p)} \times q_k)(dy_{0:k})\right) \delta_{x_{0:k}}(dx'_{0:k}) \,. \tag{A.15}$$

Lemma A.1.1. *Given Assumption 1, $K_0^{draw}(x_0, dx'_0)$ is an independent Metropolis kernel, uniformly ergodic of invariant distribution $\pi_0(dx_1)$.*

Proof. From eq. (A.12), we see that K_0^{draw} is an independent Metropolis kernel with target distribution π_0 and proposal q_0. If Assumption 1 is satisfied, uniform ergodicity follows from Corollary 4 in [365]. □

Proposition 1. *Given Assumption 1, for any $k > 0$, $K_k^{draw}(x_{0:k}, dx'_{0:k})$ is uniformly ergodic of invariant distribution*

$$\breve{\pi}_k^{(N_p)}(dx_{0:k}) = \frac{\pi_{k/k-1}(x_{0:k-1}) \cdot (\widehat{\pi}_{k-1}^{(N_p)} \times \overline{\pi}_k)(dx_{0:k})}{\widehat{\pi}_{k-1}^{(N_p)}(\pi_{k/k-1})} \,, \tag{A.16}$$

where $\overline{\pi}_k(x_{0:k-1}, dx_k)$ and $\pi_{k/k-1}(x_{0:k-1})$ are defined by eq. (A.10) *and* (A.11)*, respectively.*

Proof. From eq. (A.13), we see that K_k^{draw} is an independent Metropolis kernel with target distribution $\breve{\pi}_k^{(N_p)}$ and proposal $\widehat{\pi}_{k-1}^{(N_p)} \times q_k$. From eq. (2.61), (2.62), (A.10) and (A.11) we have

$$\begin{aligned}
\breve{\pi}_k^{(N_p)}(x_{0:k}) &= \frac{\pi_{k/k-1}(x_{0:k-1})\overline{\pi}_k(x_{0:k-1}, x_k)\widehat{\pi}_{k-1}^{(N_p)}(x_{0:k-1})}{\int_{E_k} \pi_{k/k-1}(x_{0:k-1})\overline{\pi}_k(x_{0:k-1}, x_k)\widehat{\pi}_{k-1}^{(N_p)}(x_{0:k-1})dx_{0:k}} \,, \\
&= \frac{\pi_{k/k-1}(x_{0:k-1}) \cdot (\widehat{\pi}_{k-1}^{(N_p)} \times \overline{\pi}_k)(x_{0:k})}{\int_{E_{k-1}} \pi_{k/k-1}(x_{0:k-1})\widehat{\pi}_{k-1}^{(N_p)}(x_{0:k-1})dx_{0:k-1}} \,, \\
&= \frac{\pi_{k/k-1}(x_{0:k-1}) \cdot (\widehat{\pi}_{k-1}^{(N_p)} \times \overline{\pi}_k)(x_{0:k})}{\widehat{\pi}_{k-1}^{(N_p)}(\pi_{k/k-1})} \,.
\end{aligned} \tag{A.17}$$

□

We denote the cumulative MCMC kernel of all the refinement steps by $K_k^{refine} : E_k \times \mathcal{F}_k \to [0, 1]$ for $k > 0$. We note that K_k^{refine} also has the same invariant distribution $\breve{\pi}_k^{(N_p)}$, like

K_k^{draw}. We define the overall MCMC kernel for SMCMC, $K_k : E_k \times \mathcal{F}_k \to [0,1]$ for $k \geqslant 0$ as follows,

$$K_0(x_0, dx_0^{'}) := K_0^{draw}(x_0, dx_0^{'}) \,, \tag{A.18}$$

and for $k > 0$,

$$\begin{aligned} K_k(x_{0:k}, dx_{0:k}^{'}) : &= K_k^{refine} K_k^{draw}(x_{0:k}, dx_{0:k}^{'}) \,, \\ &= \int_{E_k} K_k^{refine}(y_{0:k}, dx_{0:k}^{'}) K_k^{draw}(x_{0:k}, dy_{0:k}) \,. \end{aligned} \tag{A.19}$$

Proposition 2. *For any $k \geqslant 0$, $K_k(x_{0:k}, dx_{0:k}^{'})$ is uniformly ergodic of invariant distribution $\breve{\pi}_k^{(N_p)}(dx_{0:k})$.*

Proof. For $k = 0$, the result is trivially true from the definition of K_0 in eq. (A.18) and Lemma A.1.1. For $k > 0$, the assertion follows from application of Corollary 4 and Proposition 4 in [365] to the definition of K_k in eq. (A.19). □

Proposition 3. *Suppose $\pi \in \mathcal{P}(E)$ and $K : E \times \mathcal{F} \to [0,1]$ is an ergodic MCMC kernel of invariant distribution $\pi(dx)$. For any $f : E \to \mathbb{R}$, if $f \in \mathcal{L}^1(E, \mathcal{F}, \pi)$, $\widehat{\pi}^{(N_p)}(f)$ converges to $\pi(f)$ almost surely, irrespective of the starting point of the Markov chain $x^{(0)}$.*

Proof. This follows directly from Theorem 3 in [365]. □

A.2 Convergence of MC Estimates

Theorem A.2.1. *Given Assumption 1, for any $k \geqslant 0$, $x_{0:l}^{(0)} \in \mathcal{S}_l$ for $0 \leqslant l \leqslant k$ and $f_k \in \mathcal{L}^1(E_k, \mathcal{F}_k, \pi_k)$,*

$$\widehat{\pi}_k^{(N_p)}(f_k) \longrightarrow \pi_k(f_k) \tag{A.20}$$

almost surely, as $N_p \to \infty$.

Proof. We prove the theorem using induction over k. For $k = 0$, the theorem is trivially true which can be seen by applying Proposition 3 with Lemma A.1.1. Let us assume that the

theorem is true for $k-1$. We consider the following decomposition and examine each term individually.

$$\widehat{\pi}_k^{(N_p)}(f_k) - \pi_k(f_k) = [\widehat{\pi}_k^{(N_p)}(f_k) - \breve{\pi}_k^{(N_p)}(f_k)] + [\breve{\pi}_k^{(N_p)}(f_k) - \pi_k(f_k)]\,. \tag{A.21}$$

From eq. (A.16), we have

$$\lim_{N_p\to\infty} \breve{\pi}_k^{(N_p)}(f_k) = \frac{\lim\limits_{N_p\to\infty} \widehat{\pi}_{k-1}^{(N_p)}(\pi_{k/k-1}\bar{f}_k)}{\lim\limits_{N_p\to\infty} \widehat{\pi}_{k-1}^{(N_p)}(\pi_{k/k-1})}\,, \tag{A.22}$$

where,

$$\bar{f}_k(x_{0:k-1}) = \int_E f_k(x_{0:k-1}, x_k)\overline{\pi}_k(x_{0:k-1}, dx_k)\,. \tag{A.23}$$

We note that, from eq. (A.11),

$$\begin{aligned}\pi_{k-1}(\pi_{k/k-1}) &= \int_{E_{k-1}} \pi_k(dx_{0:k-1})\,,\\ &= \int_{E_k} \pi_k(dx_{0:k}) = 1\,,\end{aligned} \tag{A.24}$$

and from eq. (A.23),

$$\begin{aligned}\pi_{k-1}(\pi_{k/k-1}\bar{f}_k) &= \int_{E_{k-1}} \bar{f}_k(x_{0:k-1})\pi_k(dx_{0:k-1})\,,\\ &= \int_{E_k} f_k(x_{0:k})\pi_k(dx_{0:k})\,,\\ &= \pi_k(f_k) \leqslant \pi_k(|f_k|) < \infty\,,\end{aligned} \tag{A.25}$$

because $f_k \in \mathcal{L}^1(E_k, \mathcal{F}_k, \pi_k)$. As Theorem A.2.1 holds for $k-1$, we have from eq. (A.24) and (A.25)

$$\widehat{\pi}_{k-1}^{(N_p)}(\pi_{k/k-1}) \longrightarrow 1\,, \tag{A.26}$$

and

$$\widehat{\pi}_{k-1}^{(N_p)}(\pi_{k/k-1}\bar{f}_k) \longrightarrow \pi_k(f_k)\,, \tag{A.27}$$

almost surely, as $N_p \to \infty$ for any $x_{0:l}^{(0)} \in \mathcal{S}_l$ for $0 \leqslant l \leqslant k-1$. Applying eq. (A.22), this implies that

$$\lim_{N_p\to\infty} \breve{\pi}_k^{(N_p)}(f_k) - \pi_k(f_k) = 0\,, \tag{A.28}$$

almost surely.

In the SMCMC algorithm, at time step k, the MCMC kernel used to sample $\{x_{0:k}^i\}_{i=1}^{N_p}$ is K_k, which, from Proposition 2, is uniformly ergodic of invariant distribution $\breve{\pi}_k^{(N_p)}$. The applicability of Proposition 3 depends on the integrability of f_k with respect to $\lim_{N_p\to\infty} \breve{\pi}_k^{(N_p)}$. From (A.28), if $f_k \in \mathcal{L}^1(E_k, \mathcal{F}_k, \pi_k)$, we also have $f_k \in \mathcal{L}^1(E_k, \mathcal{F}_k, \lim_{N_p\to\infty} \breve{\pi}_k^{(N_p)})$ with probability 1, which allows us to use Proposition 3 to obtain

$$\widehat{\pi}_k^{(N_p)}(f_k) - \lim_{N_p\to\infty} \breve{\pi}_k^{(N_p)}(f_k) \longrightarrow 0\,, \tag{A.29}$$

almost surely, as $N_p \to \infty$, for any $x_{0:k}^{(0)} \in \mathcal{S}_k$. We use the equivalence in (A.28) and the almost sure convergence in (A.29) in eq. (A.21) to complete the proof of the theorem. □

A.3 Convergence of Normalizing Constants

With eq. (A.7) and (A.8), a straightforward corollary of Theorem A.2.1 follows.

Corollary A.3.0.1. *Given Assumption 1, for any $k \geqslant 0$ and $x_{0:l}^{(0)} \in \mathcal{S}_l$ for $0 \leqslant l \leqslant k$,*

$$\widehat{Z_0}^{(N_p)} \longrightarrow Z_0\,, \tag{A.30}$$

and for $k > 0$,

$$\widehat{\left(\frac{Z_k}{Z_{k-1}}\right)}^{(N_p)} \longrightarrow \frac{Z_k}{Z_{k-1}}\,, \tag{A.31}$$

almost surely, as $N_p \to \infty$.

Proof. From eq. (A.5), we have $Z_0 = q_0(w_0)$, where $w_0 \in \mathcal{L}^1(E_0, \mathcal{F}_0, q_0)$, because of Assumption 1. A straightforward application of *Kolmogorov's Strong Law of Large Numbers* (SLLN) proves the Corollary for $k = 0$.
For $k > 0$, from eq. (A.6), we have

$$\frac{Z_k}{Z_{k-1}} = (\pi_{k-1} \times q_k)(w_k) \leqslant B_k \,,$$

because of Assumption 1. As $\{x_{0:k}^{*(i)}\}_{i=1}^{N_p}$ are i.i.d samples from $\widehat{\pi}_{k-1}^{(N_p)} \times q_k$, we have, from (A.8)

$$\begin{aligned} \widehat{\left(\frac{Z_k}{Z_{k-1}}\right)}^{(N_p)} &\longrightarrow \left(\lim_{N_p \to \infty} \widehat{\pi}_{k-1}^{(N_p)} \times q_k\right)(w_k) \\ &= (\pi_{k-1} \times q_k)(w_k) = \frac{Z_k}{Z_{k-1}} \end{aligned}$$

almost surely, as $N_p \to \infty$ for any $x_{0:l}^{(0)} \in \mathcal{S}_l$ for $0 \leqslant l \leqslant k$. The first step follows from Kolmogorov's SLLN and the second step follows from Theorem A.2.1. □

Appendix B
Additional Results for Semi-supervised Node Classification

Table B.1: Accuracy (in %) of semi-supervised node classification on fixed splits. The best and the second best results in each column are shown in bold and marked with underline respectively. Higher numbers are better.

	Algorithms	5 labels	10 labels	20 labels
Cora	ChebyNet	67.9±3.1	72.7±2.4	80.4±0.7
	GCN	74.4±0.8	74.9±0.7	81.6±0.5
	GAT	73.5±2.2	74.5±1.3	81.6±0.9
	SBM-GCN	59.3±1.3	**77.3±1.2**	**82.2±0.8**
	BGCN (Copy)	75.1±1.3*	76.7±0.7*	81.4±0.6
	BGCN (MMSBM)	75.3±0.8*	76.6±0.8*	81.2±0.8
	BGCN (NP)	**76.0±1.1***	76.8±0.9*	80.3±0.6
Citeseer	ChebyNet	53.0±1.9	67.7±1.2	70.2±0.9
	GCN	55.4±1.1	65.8±1.1	70.8±0.7
	GAT	55.4±2.6	66.1±1.7	70.8±1.0
	SBM-GCN	20.8±2.0	66.3±0.6	71.7±0.1
	BGCN (Copy)	**61.4±2.3***	69.6±0.6*	71.9±0.6*
	BGCN (MMSBM)	57.3±0.8*	70.8±0.6*	72.2±0.6*
	BGCN (NP)	59.0±1.5*	**71.7±0.8***	**72.6±0.6***
Pubmed	ChebyNet	68.1±2.5	69.4±1.6	76.0±1.2
	GCN	69.7±0.5	72.8±0.5	78.9±0.3
	GAT	70.0±0.6	71.6±0.9	76.9±0.5
	SBM-GCN	64.8±0.8	71.7±0.7	**80.6±0.4**
	BGCN (Copy)	71.2±0.5*	73.6±0.5*	79.1±0.4
	BGCN (MMSBM)	70.9±0.8*	72.3±0.8	76.6±0.7
	BGCN (NP)	**73.3±0.7***	**73.9±0.9***	79.2±0.5

In Chapter 4, we perform random partitioning of the nodes in training and test sets to report the average of node classification accuracies across different splits. We conduct another experiment where the same fixed training-test split of [331] with 20 known labels per class

is used for 50 random intializations of the learnable model parameters. The fixed splits with 5 and 10 labels per class are formed by taking the first 5 and 10 labels for each class from the original split in [331]. The hyperparameters for all algorithms are set to the same values as in the experiment with random partitioning. The average classification accuracies along with their standard errors are provided in Table B.1 for this setting. We observe that, all models have considerably lower standard errors compared to the results in Table 4.2, since the variability due to random training-test splits is absent here.

Appendix C
Additional Results for Time-Series Forecasting

In Section C.1 of this chapter, we provide point and probabilistic forecasting results and visualization of confidence intervals for PeMSD4, PeMSD7, and PeMSD8 datasets. We examine the effect of the number of particles on forecasting performance in Section C.2. The effect of learnable noise variance is investigated in Section C.3. Comparison with a variational inference based approach is summarized in Section C.4.

C.1 Results for PeMSD4, PeMSD7, and PeMSD8

We summarize MAE, RMSE, and MAPE of different techniques for point forecasting task on the PeMSD4, PeMSD7, and PeMSD8 datasets in Tables C.1, C.2, and C.3 respectively. Tables C.4, C.5, and C.6 contain the results for probabilistic forecasting, reporting the CRPS, P10QL, P50QL, and P90QL. Figures C.1, C.2, and C.3 show some examples of confidence intervals, obtained from various probabilistic approaches, for PeMSD4, PeMSD7, and PeMSD8 datasets.

C.2 Effect of Number of Particles

For this experiment, we consider three different settings with varying number of particles $N_p = 1/10/50$ for testing. The model is trained using 1 particle in each case. From Table C.7, we observe that increasing the number of particles cannot improve the point forecasting accuracy significantly, whereas the results in Table C.8 show that characterization of the prediction uncertainty is improved as more particles are used to form the approximate posterior distribution of the forecasts.

C.3 Effect of Different Learnable Noise Variance at Each Node

In this experiment, we compare the proposed state-space model with different learnable noise variance at each node (parameterized by the softplus function in eq. (5.6) in Chapter 5) with fixed and uniform noise standard deviation $\gamma = 0.01/0.05/0.10$ at all nodes. Other hyperparameters and the training setup remain unchanged. The results in Table C.9 demonstrate that the learnable noise variance approach is not particularly beneficial in comparison to a uniform, fixed variance approach in most cases. However, we note that the probabilistic metrics reported in Table C.10 are the lowest for the learnable noise variance model in all cases. This suggests that different time-series in these road traffic datasets have different degrees of uncertainty which cannot be effectively modelled by the uniform, fixed noise variance approach.

C.4 Comparison with a Variational Inference (VI) Approach

Although there is no directly applicable baseline forecasting method in the literature that incorporates VI, RNNs, and GNNs, we can derive a variational approach using equivalent GNN-RNN architectures and compare it to the particle flow approach. We wish to approximate $p_\Theta(\mathbf{y}_{P+1:P+Q}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q})$. So, the ELBO is defined as follows:

$$\mathcal{L}(\Theta, \Omega) = \mathbb{E}_{q_\Omega}\left[\log p_\Theta(\mathbf{y}_{P+1:P+Q}, \mathbf{x}_{1:P}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q}) - \log q_\Omega(\mathbf{x}_{1:P}|\mathbf{y}_{1:P+Q}, \mathbf{z}_{1:P+Q})\right]. \quad \text{(C.1)}$$

Now, we approximate

$$\begin{aligned} p_\Theta(\mathbf{y}_{P+1:P+Q}, \mathbf{x}_{1:P}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P+Q}) &= \int \prod_{t=P+1}^{P+Q} \Big(p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t, \mathbf{z}_t) p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t)\Big) d\mathbf{x}_{P+1:P+Q}\,, \\ &\approx \prod_{t=P+1}^{P+Q} \left[\frac{1}{N_p}\sum_{j=1}^{N_p} p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t^j, \mathbf{z}_t)\right], \end{aligned} \quad \text{(C.2)}$$

where, in the decoder, we first sample $\mathbf{x}_t^j$ from $p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}^j, \mathbf{y}_{t-1}^j, \mathbf{z}_t)$ (for $t > P+1$) or from $p_{\psi,\sigma}(\mathbf{x}_t|\mathbf{x}_{t-1}^j, \mathbf{y}_{t-1}, \mathbf{z}_t)$ (for $t = P+1$) for $1 \leqslant j \leqslant N_p$ and then sample $\mathbf{y}_t^j$ from $p_{\phi,\gamma}(\mathbf{y}_t|\mathbf{x}_t^j, \mathbf{z}_t)$ for $1 \leqslant j \leqslant N_p$ to form the MC approximation. This decoder is initialized using the output of the encoder, i.e., we sample $\mathbf{x}_{1:P}^j$ from the inference distribution $q_\Omega(\mathbf{x}_{1:P}|\mathbf{y}_{1:P+Q}, \mathbf{z}_{1:P+Q})$ for $1 \leqslant j \leqslant N_p$, which is assumed to be factorized as follows:

$$\begin{aligned} q_\Omega(\mathbf{x}_{1:P}|\mathbf{y}_{1:P+Q}, \mathbf{z}_{1:P+Q}) &= q_\Omega(\mathbf{x}_{1:P}|\mathbf{y}_{1:P}, \mathbf{z}_{1:P})\,, \\ &= q_1(\mathbf{x}_1, \mathbf{z}_1, \rho) \prod_{t=2}^{P} q_{\psi',\sigma'}(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{z}_t)\,. \end{aligned} \tag{C.3}$$

Here, we set $q_1(\mathbf{x}_1, \mathbf{z}_1, \rho) = p_1(\mathbf{x}_1, \mathbf{z}_1, \rho)$ for simplicity and we use the same RNN architecture (i.e. AGCGRU) for $q_{\psi',\sigma'}$ and $p_{\psi,\sigma}$.

We treat ρ, σ and σ' as hyperparameters and set $\rho = 1$ and $\sigma = \sigma' = 0$. This implies that $q_{\psi',\sigma'}$ is a Dirac-delta function and the maximization of ELBO (in eq. (C.1)) using SGD (SGVI) amounts to minimization of the same cost function as defined in eq. (5.11). The only difference is that now a) we have two separate AGCGRUs for encoder and decoder and b) there is no particle flow in the forward pass. We call this model AGCGRU+VI and compare it to AGCGRU+flow. The other hyperparameters are set to the same values as for the AGCGRU+flow algorithm. From Table C.11, we observe that for comparable RNN architectures, the flow based algorithm significantly outperforms the variational inference based approach in the point forecasting task. The results in Table C.12 indicate that in the probabilistic forecasting task, both particle flow and VI approaches show comparable performance despite AGCGRU+flow having approximately half of the learnable parameters of the AGCGRU+VI model.

Table C.1: Average MAE, MAPE and RMSE for PeMSD4 dataset for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
HA	3.16	7.00	6.13
ARIMA	1.53/2.01/2.37/2.68	2.92/4.06/4.96/5.73	3.11/4.36/5.25/5.95
VAR	1.66/2.12/2.39/2.57	3.27/4.33/4.95/5.36	3.09/4.02/4.51/4.83
SVR	1.48/1.91/2.23/2.49	2.88/3.97/4.86/5.61	3.11/4.29/5.08/5.66
FNN	1.48/1.90/2.23/2.51	3.04/4.09/4.98/5.80	3.08/4.27/5.08/5.68
FC-LSTM	2.20/2.22/2.23/2.26	4.95/4.97/4.99/5.05	4.89/4.92/4.95/5.01
DCRNN	1.38/1.78/2.06/2.29	2.69/3.72/4.51/5.16	2.95/4.09/4.81/5.34
STGCN	1.42/1.85/2.14/2.39	2.82/3.92/4.71/5.34	2.94/4.03/4.70/5.21
ASTGCN	1.69/2.15/2.40/2.55	3.70/4.85/5.46/5.79	3.54/4.71/5.35/5.62
GWN	1.37/1.76/2.03/2.24	**2.67**/3.73/4.52/5.15	2.94/4.07/4.77/5.28
GMAN	1.38/**1.61**/**1.76**/**1.88**	2.80/**3.42**/**3.84**/4.18	2.98/**3.70**/**4.11**/**4.41**
AGCRN	1.41/1.67/1.84/2.01	2.88/3.55/3.99/4.40	3.04/3.83/4.33/4.73
LSGCN	1.40/1.78/2.03/2.20	2.80/3.71/4.27/4.68	**2.87**/3.90/4.50/4.89
DeepGLO	1.61/1.89/2.25/2.51	3.13/4.06/5.03/5.77	3.06/4.14/4.92/5.55
N-BEATS	1.49/1.90/2.20/2.44	2.93/4.00/4.84/5.48	3.13/4.29/5.05/5.58
FC-GAGA	1.43/1.78/1.95/2.06	2.87/3.80/4.32/4.67	3.06/4.09/4.55/4.82
DeepAR	1.51/2.01/2.38/2.68	3.06/4.41/5.45/6.25	3.11/4.27/5.04/5.60
DeepFactors	1.54/2.01/2.34/2.61	3.07/4.26/5.17/5.90	3.11/4.21/4.90/5.40
MQRNN	1.37/1.76/2.03/2.25	2.68/3.72/4.51/5.17	2.94/4.05/4.73/5.20
AGCGRU+flow	**1.35**/1.63/1.78/**1.88**	**2.67**/3.44/3.87/**4.16**	2.88/3.77/4.20/4.46

Table C.2: Average MAE, MAPE and RMSE for PeMSD7 dataset for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
HA	3.98	10.92	7.20
ARIMA	2.49/3.52/4.32/5.03	5.66/8.30/10.46/12.35	4.53/6.64/8.17/9.42
VAR	2.70/3.71/4.37/4.87	6.23/8.75/10.37/11.56	4.38/5.95/6.89/7.56
SVR	2.43/3.40/4.15/4.78	5.62/8.23/10.38/12.31	4.52/6.53/7.93/9.02
FNN	2.36/3.32/4.06/4.71	5.56/8.20/10.41/12.44	4.45/6.46/7.84/8.90
FC-LSTM	3.55/3.59/3.64/3.70	9.12/9.17/9.25/9.37	6.83/6.91/6.99/7.11
DCRNN	2.23/3.06/3.67/4.18	5.19/7.50/9.31/10.90	4.26/6.05/7.28/8.24
STGCN	2.21/2.96/3.47/3.90	5.20/7.32/8.82/10.09	4.09/5.72/6.76/7.55
ASTGCN	2.71/3.72/4.28/4.60	6.68/9.51/11.06/11.86	4.64/6.53/7.60/8.13
GWN	2.23/3.03/3.56/3.98	5.26/7.63/9.25/10.56	4.27/5.99/7.03/7.76
GMAN	2.40/2.76/**2.98**/**3.16**	5.93/6.96/7.66/**8.16**	4.74/5.57/**6.06**/**6.37**
AGCRN	2.19/2.81/3.15/3.42	5.22/7.09/8.19/9.01	4.12/5.49/6.27/6.79
LSGCN	2.23/2.99/3.50/3.95	5.22/7.18/8.40/9.37	**4.03**/5.59/6.54/7.30
DeepGLO	2.55/3.32/4.16/4.85	6.10/8.31/11.16/13.19	4.53/6.30/7.68/8.84
N-BEATS	2.44/3.34/4.02/4.57	5.75/8.30/10.31/11.94	4.55/6.51/7.84/8.80
FC-GAGA	2.22/2.85/3.18/3.36	5.32/7.09/8.00/8.51	4.29/5.77/6.46/6.82
DeepAR	2.53/3.61/4.48/5.20	6.15/9.30/12.17/14.49	4.55/6.50/7.84/8.87
DeepFactors	2.51/3.47/4.17/4.71	6.14/9.04/11.21/12.93	4.47/6.21/7.30/8.08
MQRNN	2.22/3.03/3.58/4.00	5.26/7.70/9.53/10.97	4.23/5.91/6.98/7.73
AGCGRU+flow	**2.15**/**2.70**/2.99/3.19	**5.13**/**6.75**/**7.61**/8.18	4.11/**5.46**/6.12/6.54

Table C.3: Average MAE, MAPE and RMSE for PeMSD8 dataset for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
HA	2.47	5.66	5.19
ARIMA	1.24/1.61/1.89/2.12	2.33/3.15/3.77/4.31	2.63/3.62/4.28/4.81
VAR	1.37/1.79/2.04/2.23	2.66/3.62/4.23/4.69	2.67/3.53/4.01/4.36
SVR	1.21/1.56/1.80/2.01	2.32/3.12/3.72/4.24	2.64/3.57/4.18/4.63
FNN	1.19/1.54/1.79/2.01	2.27/3.12/3.75/4.30	2.59/3.55/4.17/4.63
FC-LSTM	1.91/1.93/1.94/1.95	4.63/4.66/4.69/4.72	4.71/4.75/4.78/4.81
DCRNN	1.16/1.49/1.70/1.87	2.25/3.16/3.85/4.37	2.54/3.49/4.08/4.49
STGCN	1.22/1.56/1.79/1.98	2.49/3.43/4.06/4.48	2.67/3.65/4.22/4.59
ASTGCN	1.36/1.64/1.81/1.92	3.04/3.79/4.23/4.51	2.98/3.77/4.20/4.47
GWN	**1.11**/1.40/1.59/1.73	**2.14**/**2.94**/3.49/3.90	**2.52**/3.45/4.00/4.38
GMAN	1.23/**1.36**/**1.46**/**1.55**	2.73/3.09/**3.38**/**3.63**	3.05/3.50/**3.82**/**4.06**
AGCRN	1.16/1.39/1.53/1.67	2.49/3.10/3.50/3.84	2.67/**3.44**/3.91/4.25
LSGCN	1.21/1.54/1.75/1.89	2.56/3.44/3.95/4.30	2.71/3.64/4.14/4.46
DeepGLO	1.30/1.75/2.04/2.21	2.48/3.42/4.06/4.50	2.67/3.63/4.24/4.69
N-BEATS	1.33/1.69/1.92/2.12	2.74/3.85/4.45/4.90	2.81/3.94/4.52/4.92
FC-GAGA	1.18/1.47/1.62/1.72	2.37/3.21/3.76/4.11	2.65/3.61/4.10/4.39
DeepAR	1.25/1.61/1.87/2.10	2.53/3.40/4.08/4.67	2.67/3.59/4.17/4.61
DeepFactors	1.26/1.63/1.88/2.07	2.51/3.42/4.08/4.61	2.63/3.54/4.11/4.52
MQRNN	1.13/1.43/1.62/1.77	2.19/2.99/3.56/4.00	2.54/3.48/4.02/4.40
AGCGRU+flow	1.13/1.37/1.49/1.57	2.30/3.01/3.40/3.65	2.59/3.45/3.85/4.09

Table C.4: Average CRPS, P10QL, P50QL, and P90QL for PeMSD4 for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD4
DeepAR	1.13/1.52/1.82/2.07
DeepFactors	1.52/1.84/2.07/2.26
GRU+flow	1.14/1.50/1.75/1.95
DCGRU+flow	1.13/1.43/1.63/1.79
AGCGRU+flow	**1.08/1.32/1.46/1.56**
	P10QL(%) (15/ 30/ 45/ 60 min)
DeepAR	1.37/1.96/2.45/2.86
DeepFactors	2.13/2.61/3.01/3.34
MQRNN	**0.95/1.18/1.31/1.40**
GRU+flow	1.36/1.87/2.25/2.56
DCGRU+flow	1.33/1.75/2.06/2.30
AGCGRU+flow	1.28/1.62/1.82/1.97
	P50QL(%) (15/ 30/ 45/ 60 min)
DeepAR	2.37/3.15/3.73/4.20
DeepFactors	2.42/3.15/3.68/4.10
MQRNN	2.15/2.77/3.19/3.53
GRU+flow	2.16/2.76/3.17/3.50
DCGRU+flow	2.16/2.69/3.01/3.26
AGCGRU+flow	**2.11/2.55/2.79/2.94**
	P90QL(%) (15/ 30/ 45/ 60 min)
DeepAR	1.10/1.45/1.67/1.84
DeepFactors	1.98/2.24/2.39/2.50
MQRNN	1.22/1.68/2.03/2.32
GRU+flow	1.11/1.43/1.63/1.77
DCGRU+flow	1.10/1.34/1.50/1.61
AGCGRU+flow	**1.05/1.26/1.37/1.45**

Table C.5: Average CRPS, P10QL, P50QL, and P90QL for PeMSD7 for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD7
DeepAR	1.92/2.78/3.44/3.99
DeepFactors	2.35/3.00/3.48/3.87
GRU+flow	1.88/2.61/3.09/3.46
DCGRU+flow	1.85/2.51/2.95/3.27
AGCGRU+flow	**1.73/2.18/2.43/2.58**
	P10QL(%) (15/ 30/ 45/ 60 min)
DeepAR	2.56/3.90/4.92/5.78
DeepFactors	3.49/4.53/5.46/6.26
MQRNN	**1.70/2.20/2.47/2.66**
GRU+flow	2.50/3.57/4.29/4.85
DCGRU+flow	2.41/3.35/3.97/4.43
AGCGRU+flow	2.27/2.97/3.36/3.60
	P50QL(%) (15/ 30/ 45/ 60 min)
DeepAR	4.35/6.21/7.70/8.95
DeepFactors	4.31/5.97/7.16/8.10
MQRNN	3.82/5.21/6.16/6.88
GRU+flow	3.84/5.19/6.10/6.81
DCGRU+flow	3.77/4.94/5.66/6.20
AGCGRU+flow	**3.70/4.65/5.14/5.49**
	P90QL(%) (15/ 30/ 45/ 60 min)
DeepAR	2.13/3.03/3.65/4.08
DeepFactors	3.22/3.70/3.97/4.14
MQRNN	2.19/3.12/3.78/4.30
GRU+flow	2.02/2.74/3.16/3.44
DCGRU+flow	2.00/2.62/3.01/3.28
AGCGRU+flow	**1.83/2.25/2.48/2.62**

Table C.6: Average CRPS, P10QL, P50QL, and P90QL for PeMSD8 for 15/30/45/60 minutes horizons. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better.

Algorithm	PeMSD8
DeepAR	0.94/1.24/1.46/1.64
DeepFactors	1.26/1.51/1.69/1.83
GRU+flow	0.95/1.23/1.42/1.57
DCGRU+flow	0.94/1.18/1.35/1.47
AGCGRU+flow	**0.90/1.10/1.20/1.28**
	P10QL(%) (15/ 30/ 45/ 60 min)
DeepAR	1.14/1.59/1.93/2.24
DeepFactors	1.77/2.17/2.49/2.76
MQRNN	**0.77/0.94/1.04/1.10**
GRU+flow	1.12/1.52/1.80/2.04
DCGRU+flow	1.10/1.43/1.67/1.87
AGCGRU+flow	1.10/1.43/1.61/1.73
	P50QL(%) (15/ 30/ 45/ 60 min)
DeepAR	1.97/2.52/2.94/3.30
DeepFactors	1.97/2.55/2.95/3.25
MQRNN	1.77/2.24/2.54/2.77
GRU+flow	**1.76**/2.21/2.49/2.72
DCGRU+flow	1.83/2.25/2.49/2.66
AGCGRU+flow	1.78/**2.15/2.34/2.46**
	P90QL(%) (15/ 30/ 45/ 60 min)
DeepAR	0.93/1.22/1.40/1.53
DeepFactors	1.62/1.82/1.93/1.99
MQRNN	0.99/1.34/1.59/1.80
GRU+flow	0.93/1.18/1.33/1.44
DCGRU+flow	0.93/1.13/1.25/1.34
AGCGRU+flow	**0.87/1.01/1.09/1.14**

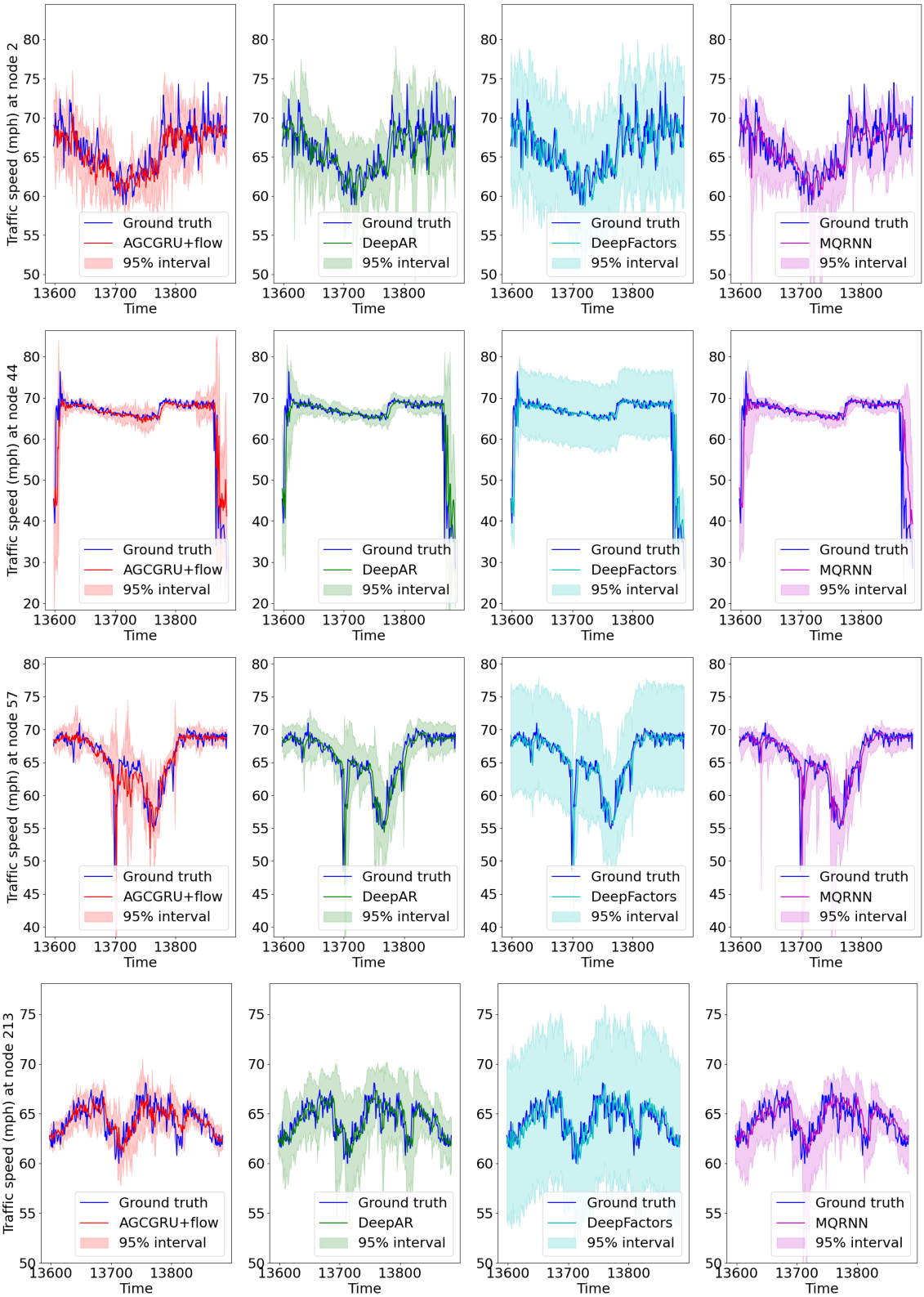


Figure C.1: 15 minutes ahead predictions from the probabilistic forecasting algorithms with confidence intervals at nodes 2, 44, 57, and 213 of PeMSD4 dataset for the first day in the test set. The proposed AGCGRU+flow algorithm provides tighter confidence interval than its competitors in most cases, which leads to lower quantile error.

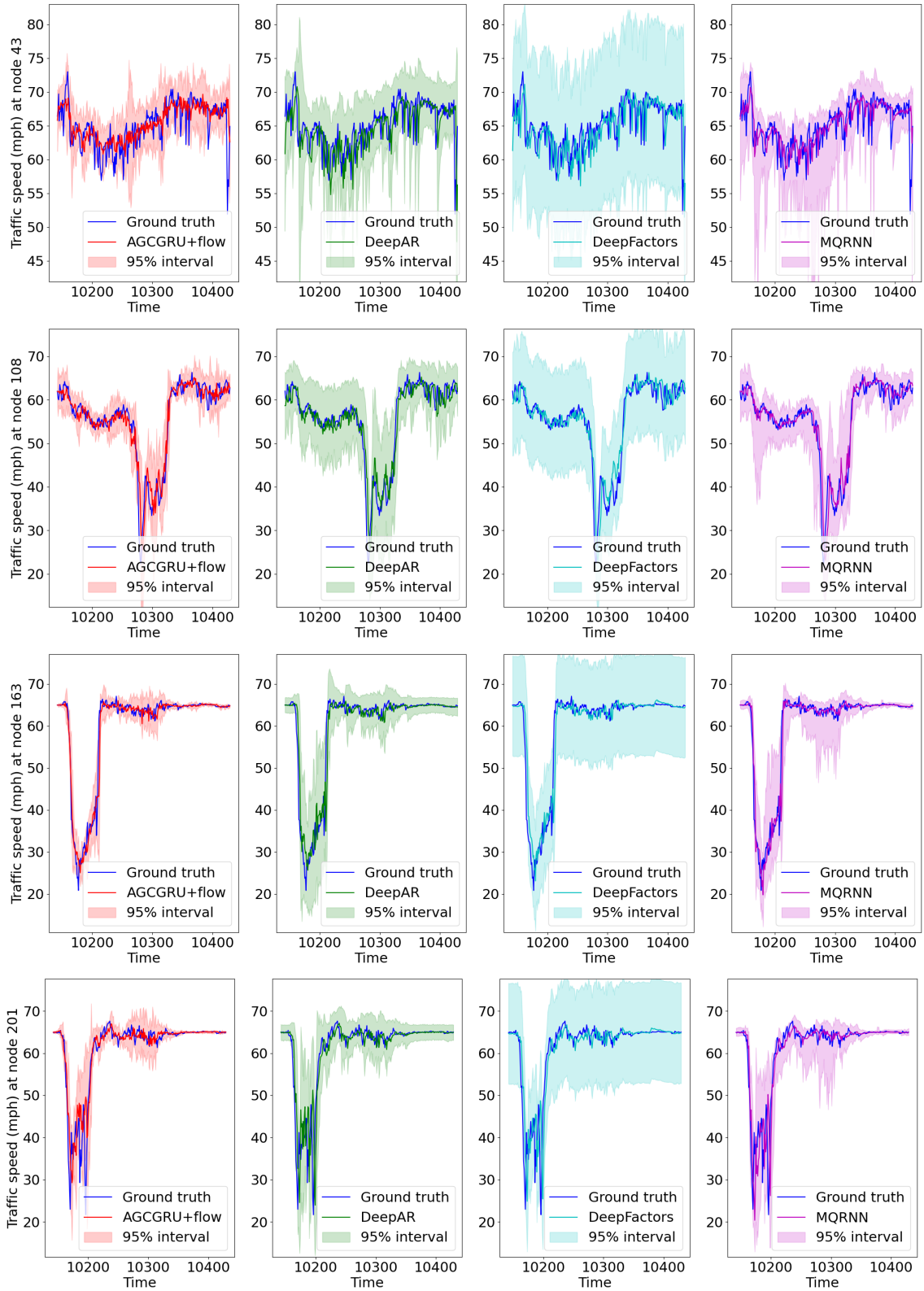


Figure C.2: 15 minutes ahead predictions from the probabilistic forecasting algorithms with confidence intervals at nodes 43, 108, 163, and 201 of PeMSD7 dataset for the first day in the test set. The proposed AGCGRU+flow algorithm provides tighter confidence interval than its competitors in most cases, which leads to lower quantile error.

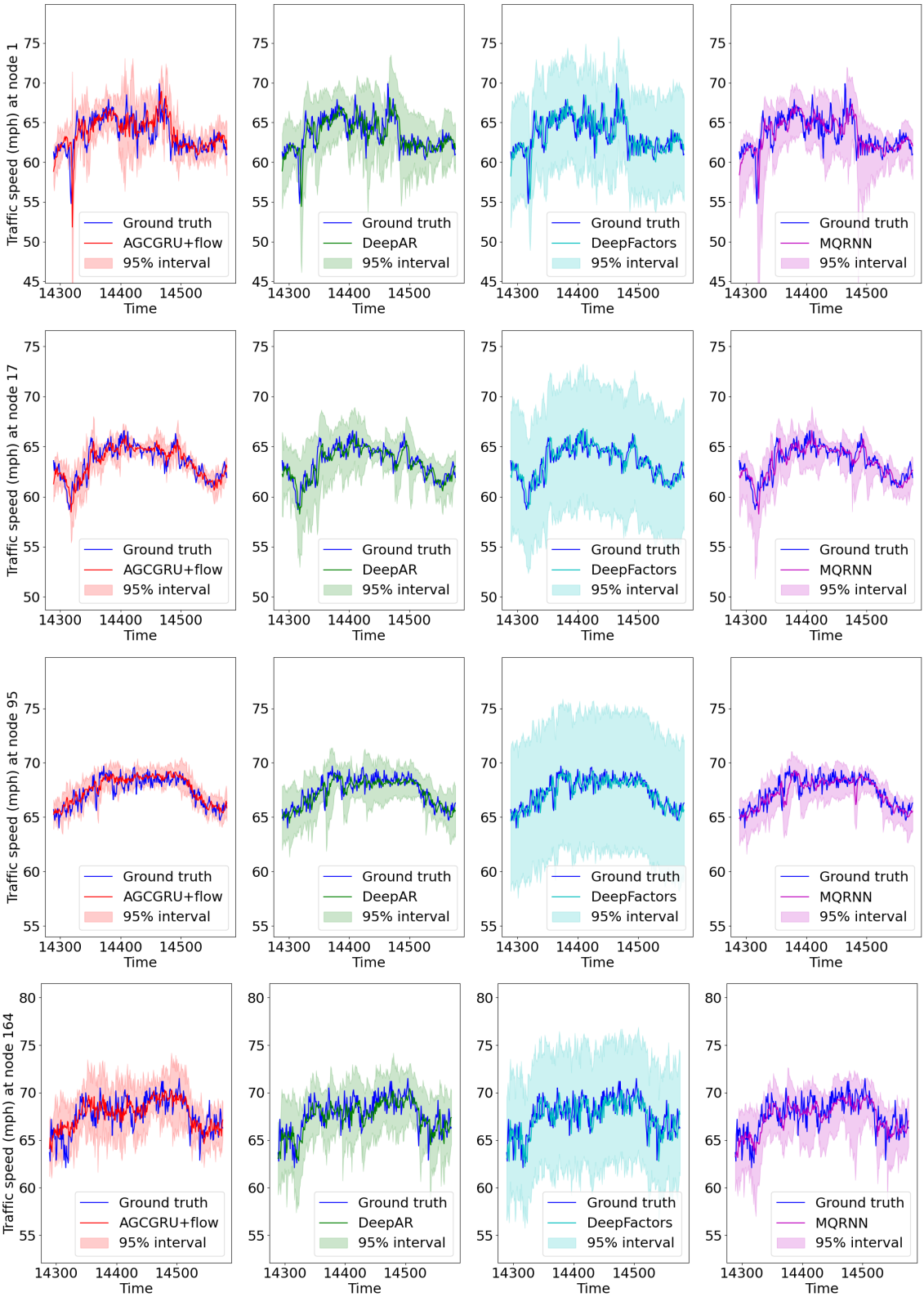


Figure C.3: 15 minutes ahead predictions from the probabilistic forecasting algorithms with confidence intervals at nodes 1, 17, 95, and 164 of PeMSD8 dataset for the first day in the test set. The proposed AGCGRU+flow algorithm provides tighter confidence interval than its competitors in most cases, which leads to lower quantile error.

Table C.7: Average MAE, MAPE, and RMSE for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow with different number of particles. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow ($N_p = 1$)	13.82/14.87/15.60/16.08	14.04/14.78/15.36/15.82	22.33/24.41/25.70/26.54
AGCGRU+flow ($N_p = 10$)	**13.79**/**14.84**/**15.58**/**16.06**	**14.01**/14.75/15.34/15.80	22.08/24.26/**25.55**/26.43
AGCGRU+flow ($N_p = 50$)	**13.79**/**14.84**/**15.58**/**16.06**	**14.01**/**14.74**/**15.33**/**15.79**	**22.02**/**24.20**/**25.55**/**26.42**
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow ($N_p = 1$)	1.35/1.63/1.78/1.88	2.68/3.45/3.89/4.18	2.89/3.78/4.22/4.47
AGCGRU+flow ($N_p = 10$)	1.35/1.63/1.78/1.88	**2.67**/**3.44**/**3.87**/**4.16**	**2.88**/**3.77**/**4.20**/4.46
AGCGRU+flow ($N_p = 50$)	1.35/1.63/1.78/1.88	**2.67**/**3.44**/**3.87**/**4.16**	**2.88**/**3.77**/**4.20**/**4.45**
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow ($N_p = 1$)	2.16/2.71/3.00/3.20	5.14/6.77/7.63/8.20	4.12/5.47/6.14/6.56
AGCGRU+flow ($N_p = 10$)	**2.15**/**2.70**/**2.99**/**3.19**	5.13/**6.75**/**7.61**/**8.18**	**4.11**/**5.46**/**6.12**/**6.54**
AGCGRU+flow ($N_p = 50$)	**2.15**/**2.70**/**2.99**/**3.19**	**5.12**/**6.75**/**7.61**/**8.18**	**4.11**/**5.46**/**6.12**/**6.54**
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow ($N_p = 1$)	1.14/1.38/1.50/1.57	2.31/3.02/3.41/3.67	2.60/3.46/3.87/4.11
AGCGRU+flow ($N_p = 10$)	**1.13**/**1.37**/**1.49**/1.57	**2.30**/**3.01**/**3.40**/**3.65**	**2.59**/3.45/**3.85**/**4.09**
AGCGRU+flow ($N_p = 50$)	**1.13**/**1.37**/**1.49**/1.57	**2.30**/**3.01**/**3.40**/**3.65**	**2.59**/**3.44**/**3.85**/**4.09**

Table C.8: Average CRPS, P10QL, and P90QL for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow with different number of particles. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow ($N_p = 1$)	19.34/20.44/21.24/21.80	11.79/12.80/13.46/13.91	10.46/10.72/10.98/11.18
AGCGRU+flow ($N_p = 10$)	10.53/11.39/12.03/12.47	4.01/4.44/4.76/4.97	4.06/4.38/4.63/4.82
AGCGRU+flow ($N_p = 50$)	**10.02/10.86/11.49/11.92**	**3.67/4.05/4.33/4.53**	**3.83/4.15/4.41/4.59**
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow ($N_p = 1$)	1.95/2.34/2.58/2.73	3.11/3.75/4.16/4.47	3.00/3.59/3.92/4.10
AGCGRU+flow ($N_p = 10$)	1.08/1.32/1.46/1.56	1.28/1.62/1.82/1.97	1.05/1.26/1.37/1.45
AGCGRU+flow ($N_p = 50$)	**1.03/1.26/1.40/1.49**	**1.21/1.54/1.73/1.87**	**0.98/1.17/1.27/1.35**
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow ($N_p = 1$)	3.18/3.95/4.35/4.61	5.57/6.96/7.67/8.15	5.38/6.63/7.29/7.69
AGCGRU+flow ($N_p = 10$)	1.73/2.18/2.43/2.58	2.27/2.97/3.36/3.60	1.83/2.25/2.48/2.62
AGCGRU+flow ($N_p = 50$)	**1.64/2.09/2.32/2.47**	**2.16/2.83/3.20/3.44**	**1.71/2.10/2.31/2.45**
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow ($N_p = 1$)	1.63/1.90/2.07/2.18	2.73/3.28/3.63/3.87	2.38/2.68/2.86/2.98
AGCGRU+flow ($N_p = 10$)	0.90/1.10/1.20/1.28	1.10/1.43/1.61/1.73	0.87/1.01/1.09/1.14
AGCGRU+flow ($N_p = 50$)	**0.86/1.05/1.16/1.22**	**1.04/1.35/1.52/1.63**	**0.83/0.95/1.03/1.08**

Table C.9: Average MAE, MAPE, and RMSE for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow with learnable and fixed noise variance settings. The best result in each column is shown in bold. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow (learnable)	13.79/14.84/15.58/16.06	**14.01**/**14.75**/**15.34**/**15.80**	22.08/24.26/25.55/26.43
AGCGRU+flow ($\gamma = 0.01$)	**13.68**/**14.75**/**15.49**/**16.02**	14.57/15.37/16.02/16.57	**21.74**/**23.95**/**25.27**/**26.21**
AGCGRU+flow ($\gamma = 0.05$)	13.96/15.05/15.76/16.25	15.87/16.66/17.23/17.62	22.08/24.33/25.64/26.54
AGCGRU+flow ($\gamma = 0.10$)	13.86/14.91/15.68/16.17	14.42/15.20/15.87/16.39	22.04/24.25/25.60/26.41
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow (learnable)	**1.35**/**1.63**/**1.78**/**1.88**	**2.67**/**3.44**/**3.87**/**4.16**	**2.88**/3.77/**4.20**/**4.46**
AGCGRU+flow ($\gamma = 0.01$)	**1.35**/**1.63**/1.79/1.89	2.68/3.45/3.89/4.20	**2.88**/3.77/**4.20**/4.47
AGCGRU+flow ($\gamma = 0.05$)	1.36/1.65/1.80/1.91	2.69/3.47/3.91/4.21	**2.88**/**3.76**/**4.20**/**4.46**
AGCGRU+flow ($\gamma = 0.10$)	1.36/1.65/1.80/1.90	2.70/3.47/3.89/4.18	2.92/3.81/4.24/4.49
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow (learnable)	2.15/2.70/2.99/3.19	5.13/6.75/7.61/8.18	4.11/5.46/6.12/6.54
AGCGRU+flow ($\gamma = 0.01$)	**2.14**/**2.69**/**2.98**/**3.16**	**5.07**/**6.66**/**7.47**/**8.00**	4.10/5.43/6.09/6.49
AGCGRU+flow ($\gamma = 0.05$)	2.16/2.71/3.00/3.20	5.13/6.74/7.61/8.19	**4.09**/**5.41**/**6.06**/**6.48**
AGCGRU+flow ($\gamma = 0.10$)	2.16/2.73/3.01/3.20	5.15/6.77/7.62/8.15	4.12/5.48/6.15/6.54
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow (learnable)	1.13/**1.37**/**1.49**/**1.57**	2.30/3.01/3.40/3.65	2.59/3.45/3.85/4.09
AGCGRU+flow ($\gamma = 0.01$)	1.13/**1.37**/**1.49**/**1.57**	2.31/3.03/3.44/3.71	2.60/3.43/3.84/4.09
AGCGRU+flow ($\gamma = 0.05$)	1.13/**1.37**/**1.49**/**1.57**	**2.26**/**2.95**/**3.35**/**3.62**	**2.53**/**3.34**/**3.75**/**4.01**
AGCGRU+flow ($\gamma = 0.10$)	1.13/1.38/1.51/1.60	2.31/3.04/3.49/3.80	2.57/3.41/3.86/4.14

Table C.10: Average CRPS, P10QL, and P90QL for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow with learnable and fixed noise variance settings. The best result in each column is shown in bold. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow (learnable)	**10.53/11.39/12.03/12.47**	**4.01/4.44/4.76/4.97**	**4.06/4.38/4.63/4.82**
AGCGRU+flow ($\gamma = 0.01$)	12.83/13.90/14.63/15.17	7.26/8.10/8.46/8.77	6.68/7.08/7.55/7.86
AGCGRU+flow ($\gamma = 0.05$)	11.58/12.61/13.28/13.74	5.78/6.52/6.99/7.25	5.14/5.54/5.81/6.06
AGCGRU+flow ($\gamma = 0.10$)	13.14/14.18/14.95/15.43	7.79/8.57/9.22/9.53	6.64/7.05/7.28/7.53
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow (learnable)	**1.08/1.32/1.46/1.56**	**1.28/1.62/1.82/1.97**	**1.05/1.26/1.37/1.45**
AGCGRU+flow ($\gamma = 0.01$)	1.28/1.55/1.70/1.81	2.09/2.58/2.87/3.08	1.74/2.08/2.26/2.38
AGCGRU+flow ($\gamma = 0.05$)	1.19/1.47/1.62/1.72	1.82/2.30/2.57/2.77	1.48/1.84/2.04/2.15
AGCGRU+flow ($\gamma = 0.10$)	1.32/1.60/1.75/1.85	2.19/2.68/2.95/3.15	1.84/2.23/2.43/2.54
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow (learnable)	**1.73/2.18/2.43/2.58**	**2.27/2.97/3.36/3.60**	**1.83/2.25/2.48/2.62**
AGCGRU+flow ($\gamma = 0.01$)	2.02/2.55/2.82/3.01	3.59/4.57/5.05/5.35	3.00/3.77/4.22/4.54
AGCGRU+flow ($\gamma = 0.05$)	1.90/2.42/2.70/2.90	3.18/4.20/4.76/5.15	2.56/3.27/3.65/3.91
AGCGRU+flow ($\gamma = 0.10$)	2.09/2.65/2.94/3.12	3.80/4.87/5.41/5.77	3.18/4.04/4.47/4.73
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow (learnable)	**0.90/1.10/1.20/1.28**	**1.10/1.43/1.61/1.73**	**0.87/1.01/1.09/1.14**
AGCGRU+flow ($\gamma = 0.01$)	1.07/1.29/1.41/1.49	1.81/2.29/2.56/2.75	1.35/1.57/1.67/1.73
AGCGRU+flow ($\gamma = 0.05$)	1.00/1.23/1.35/1.43	1.58/2.04/2.31/2.50	1.21/1.43/1.52/1.58
AGCGRU+flow ($\gamma = 0.10$)	1.10/1.34/1.47/1.56	1.88/2.41/2.72/2.93	1.47/1.71/1.81/1.87

Table C.11: Average MAE, MAPE, and RMSE for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow and AGCGRU+VI. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**13.79/14.84/15.58/16.06**	**14.01/14.75/15.34/15.80**	**22.08/24.26/25.55/26.43**
AGCGRU+VI	15.08/16.10/16.83/17.53	15.26/16.10/16.74/17.43	26.17/28.02/29.13/30.17
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**1.35/1.63/1.78/1.88**	**2.67/3.44/3.87/4.16**	**2.88/3.77/4.20/4.46**
AGCGRU+VI	1.46/1.76/1.94/2.06	2.94/3.73/4.20/4.52	2.97/3.78/4.22/4.48
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**2.15/2.70/2.99/3.19**	**5.13/6.75/7.61/8.18**	**4.11/5.46**/6.12/6.54
AGCGRU+VI	2.33/2.92/3.23/3.45	5.59/7.26/8.16/8.78	4.22/5.48/**6.10/6.50**
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	MAE	MAPE(%)	RMSE
AGCGRU+flow	**1.13/1.37/1.49/1.57**	**2.30/3.01/3.40/3.65**	**2.59/3.45/3.85/4.09**
AGCGRU+VI	1.29/1.52/1.65/1.74	2.94/3.51/3.86/4.10	2.96/3.59/3.94/4.17

Table C.12: Average CRPS, P10QL, and P90QL for PeMSD3, PeMSD4, PeMSD7, and PeMSD8 for 15/30/45/60 minutes horizons for AGCGRU+flow and AGCGRU+VI. Lower numbers are better.

Algorithm	PeMSD3 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	**10.53**/**11.39**/**12.03**/**12.47**	**4.01**/**4.44**/**4.76**/**4.97**	**4.06**/**4.38**/**4.63**/**4.82**
AGCGRU+VI	11.00/11.80/12.38/12.94	4.14/4.53/4.82/5.10	4.27/4.58/4.81/5.02
Algorithm	PeMSD4 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	1.08/1.32/1.46/1.56	1.28/1.62/1.82/1.97	1.05/1.26/1.37/1.45
AGCGRU+VI	1.08/**1.31**/**1.45**/**1.54**	**1.26**/**1.59**/**1.79**/**1.93**	**1.04**/**1.25**/**1.36**/1.45
Algorithm	PeMSD7 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	1.73/2.18/2.43/**2.58**	2.27/2.97/**3.36**/**3.60**	1.83/2.25/2.48/**2.62**
AGCGRU+VI	**1.72**/2.18/**2.42**/2.60	**2.25**/2.97/3.39/3.66	**1.80**/**2.24**/**2.47**/2.63
Algorithm	PeMSD8 (15/ 30/ 45/ 60 min)		
	CRPS	P10QL(%)	P90QL(%)
AGCGRU+flow	**0.90**/**1.10**/**1.20**/**1.28**	**1.10**/**1.43**/**1.61**/**1.73**	**0.87**/**1.01**/**1.09**/**1.14**
AGCGRU+VI	0.95/1.13/1.24/1.31	1.15/1.44/1.62/1.76	0.90/1.03/1.10/1.15

Bibliography

[1] G. J. Kerns and G. J. Székely, "Definetti's theorem for abstract finite exchangeable sequences," *J. Theoretical Probab.*, vol. 19, no. 3, pp. 589–608, Jul. 2006.

[2] D. V. Lindley, "The future of statistics: A Bayesian 21st century," *Adv. Appl. Probab.*, vol. 7, pp. 106–115, 1975.

[3] G. Casella, C. P. Robert, and M. T. Wells, "Generalized accept-reject sampling schemes," in *A Festschrift for Herman Rubin*. Institute of Mathematical Statistics, Jan. 2004, pp. 342–347.

[4] T. Kloek and H. K. van Dijk, "Bayesian estimates of equation system parameters: An application of integration by monte carlo," *Econometrica*, vol. 46, no. 1, pp. 1–19, Jan. 1978.

[5] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970.

[6] N. Tishby, E. Levin, and S. Solla, "Consistent inference of probabilities in layered networks: Predictions and generalization," in *Proc. Int. Joint Conf. Neural Networks*, Washington, WA, USA, Jun. 1989.

[7] D. J. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Comp.*, vol. 4, no. 3, pp. 448–472, May 1992.

[8] R. M. Neal, "Bayesian learning via stochastic dynamics," in *Proc. Adv. Neural Info. Process. Syst.*, San Francisco, CA, USA, Dec. 1992, pp. 475–482.

[9] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson, "What are Bayesian neural network posteriors really like?" in *Proc. Int. Conf. Machine Learning*, Virtual, Jul. 2021.

[10] R. Martinez-Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet, "A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot," *Autonomous Robots*, vol. 27, no. 2, pp. 93–103, Aug. 2009.

[11] S. Nannuru, Y. Li, Y. Zeng, M. Coates, and B. Yang, "Radio-frequency tomography for passive indoor multitarget tracking," *IEEE Trans. Mob. Comput.*, vol. 12, no. 12, pp. 2322–2333, Dec. 2013.

[12] P. J. van Leeuwen, H. R. Künsch, L. Nerger, R. Potthast, and S. Reich, "Particle filters for high-dimensional geoscience applications: A review," *Quarterly J. Royal Meteor. Soc.*, vol. 145, no. 723, pp. 2335–2365, Apr. 2019.

[13] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proc. F Radar and Signal Process.*, vol. 140, no. 2, Apr. 1993, pp. 107–113.

[14] A. Beskos, D. Crisan, and A. Jasra, "On the stability of sequential Monte Carlo methods in high dimensions," *Ann. Appl. Prob.*, vol. 24, no. 4, pp. 1396–1445, Aug. 2014.

[15] F. Septier and G. W. Peters, "Langevin and Hamiltonian based sequential MCMC for efficient Bayesian filtering in high-dimensional spaces," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 2, pp. 312–327, Mar. 2016.

[16] P. M. Djurić, T. Lu, and M. F. Bugallo, "Multiple particle filtering," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, vol. 3, Honolulu, HI, USA, Apr. 2007, pp. 1181–1184.

[17] P. M. Djurić and M. F. Bugallo, "Particle filtering for high-dimensional systems," in *Proc. IEEE Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.*, St. Martin, France, Dec. 2013, pp. 352–355.

[18] A. Beskos, D. Crisan, A. Jasra, K. Kamatani, and Y. Zhou, "A stable particle filter in high-dimensions," *ArXiv e-prints: arXiv 1412.3501*, Dec. 2014.

[19] P. Rebeschini and R. van Handel, "Can local particle filters beat the curse of dimensionality?" *Ann. Appl. Probab.*, vol. 25, no. 5, pp. 2809–2866, Oct. 2015.

[20] F. Daum and J. Huang, "Nonlinear filters with log-homotopy," in *Proc. SPIE Signal and Data Process. Small Targets*, San Diego, CA, USA, Sep. 2007, p. 669918.

[21] F. Daum, J. Huang, and A. Noushin, "Exact particle flow for nonlinear filters," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Orlando, FL, USA, Apr. 2010, p. 769704.

[22] S. Reich, "A guided sequential Monte Carlo method for the assimilation of data into stochastic dynamical systems," in *Recent Trends in Dynamical Systems.* Basel, Switzerland: Springer, 2013, vol. 35, pp. 205–220.

[23] P. Bunch and S. Godsill, "Approximations of the optimal importance density using gaussian particle flow importance sampling," *J. Amer. Statist. Assoc.*, vol. 111, no. 514, pp. 748–762, Aug. 2016.

[24] F. E. de Melo, S. Maskell, M. Fasiolo, and F. Daum, "Stochastic particle flow for nonlinear high-dimensional filtering problems," *ArXiv e-prints: arXiv 1511.01448*, Nov. 2015.

[25] J. Heng, A. Doucet, and Y. Pokern, "Gibbs flow for approximate transport with applications to Bayesian computation," *ArXiv e-prints: arXiv 1509.08787*, Sep. 2015.

[26] Y. Li and M. Coates, "Particle filtering with invertible particle flow," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4102–4116, Aug. 2017.

[27] F. Septier, S. K. Pang, A. Carmi, and S. Godsill, "On MCMC-based particle methods for Bayesian filtering: Application to multitarget tracking," in *Proc. IEEE Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.*, Aruba, The Netherlands, Dec. 2009, pp. 360–363.

[28] D. Alspach and H. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximations," *IEEE Trans. Automatic Control*, vol. 17, no. 4, pp. 439–448, Aug. 1972.

[29] B. Anderson and J. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[30] J. H. Kotecha and P. M. Djuric, "Gaussian sum particle filtering," *IEEE Trans. Signal Process.*, vol. 51, no. 10, pp. 2602–2612, Oct. 2003.

[31] M. A. Khan, M. Ulmke, and W. Koch, "A log homotopy based particle flow solution for mixture of Gaussian prior densities," in *Proc. IEEE Int. Conf. Multisensor Fusion and Integration for Intell. Syst.*, Munich, Germany, Sep. 2016, pp. 546–551.

[32] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE Trans. Neural Networks*, vol. 9, no. 5, pp. 768–786, Nov. 1998.

[33] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. IEEE Int. Joint Conf. Neural Networks*, vol. 2, Montreal, Canada, Aug. 2005, pp. 729–734.

[34] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[35] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Machine Learning*, Sydney, Australia, Aug. 2017, pp. 1263–1272.

[36] W. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *ArXiv e-prints: arXiv 1709.05584*, 2017.

[37] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learning Representations*, Toulon, France, Apr. 2017.

[38] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learning Representations*, Vancouver, Canada, May 2018.

[39] T. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. Bayesian Deep Learning Workshop, Adv. Neural Info. Process. Syst.*, Barcelona, Spain, Dec. 2016.

[40] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *Proc. Int. Conf. Machine Learning*, Long Beach, CA, USA, Jun. 2019, pp. 2434–2444.

[41] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2019, pp. 3276–3285.

[42] R. Wang, S. Mou, X. Wang, W. Xiao, Q. Ju, C. Shi, and X. Xie, "Graph structure estimation neural networks," in *Proc. Int. Conf. World Wide Web*, 2021, pp. 342–353.

[43] P. Elinas, E. V. Bonilla, and L. Tiao, "Variational inference for graph convolutional networks in the absence of graph data and adversarial settings," in *Proc. Adv. Neural Info. Process. Syst.*, vol. 33, Virtual, Dec. 2020, pp. 18 648–18 660.

[44] B. Jiang, Z. Zhang, J. Tang, and B. Luo, "Graph optimized convolutional networks," *ArXiv e-prints: arXiv 1904.11883*, Apr. 2019.

[45] S. Wan, S. Pan, J. Yang, and C. Gong, "Contrastive and generative graph convolutional networks for graph-based semi-supervised learning," in *Proc. AAAI Conf. Artificial*

Intell., Virtual, Feb. 2021.

[46] G. U. Yule, "On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers," *Philosophical Trans. Royal Soc. London. Ser. A*, vol. 226, pp. 267–298, Feb. 1927.

[47] S. Makridakis and M. Hibon, "ARMA models and the box-Jenkins methodology," *J. Forecasting*, vol. 16, no. 3, pp. 147–163, Jan. 1997.

[48] R. Sen, H.-F. Yu, and I. S. Dhillon, "Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2019, pp. 4837–4846.

[49] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting," in *Proc. Int. Conf. Learning Representations*, Virtual, May 2020.

[50] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learning Representations*, Vancouver, Canada, May 2018.

[51] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," in *Proc. Adv. Neural Info. Process. Syst.*, Virtual, Dec. 2020.

[52] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *Int. J. Forecasting*, vol. 36, no. 3, pp. 1181 – 1191, Sep. 2020.

[53] Y. Wang, A. Smola, D. Maddix, J. Gasthaus, D. Foster, and T. Januschowski, "Deep factors for forecasting," in *Proc. Int. Conf. Machine Learning*, Long Beach, California, USA, Jun 2019.

[54] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2018.

[55] Y. Li, S. Pal, and M. Coates, "Invertible particle flow-based sequential MCMC with extension to Gaussian mixture noise models," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2499–2512, May 2019.

[56] Y. Zhang, S. Pal, M. Coates, and D. Üstebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proc. AAAI Conf. Artificial Intell.*, Honolulu, HI, USA, Feb. 2019, pp. 5829–5836.

[57] S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, "Non-parametric graph learning for Bayesian graph neural networks," in *Proc. Conf. Uncertainty in Artificial Intell.*, Virtual, Aug. 2020.

[58] S. Pal, L. Ma, Y. Zhang, and M. Coates, "RNN with particle flow for probabilistic spatio-temporal forecasting," in *Proc. Int. Conf. Machine Learning*, Virtual, Jul. 2021.

[59] S. Pal and M. Coates, "Gaussian sum particle flow filter," in *Proc. IEEE Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.*, Curacao, The Netherlands, Dec. 2017, pp. 1–5.

[60] ——, "Particle flow particle filter for Gaussian mixture noise models," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, Calgary, Canada, Apr. 2018, pp. 4249–4253.

[61] Y. Li and M. Coates, "Sequential MCMC with invertible particle flow," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, New Orleans, LA, USA, Mar. 2017.

[62] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Info. Process. Syst.*, Barcelona, Spain, Dec. 2016, pp. 3844–3852.

[63] D. Creal and R. Tsay, "High dimensional dynamic stochastic copula models," *J. Econometrics*, vol. 189, no. 2, pp. 335–345, Dec. 2015.

[64] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

[65] G. Welch and G. Bishop, "An introduction to the Kalman filter," Univ. North Carolina Chapel Hill, USA, Tech. Rep., 1995.

[66] J. E. Handschin and D. Q. Mayne, "Monte carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering," *Int. J. Control*, vol. 9, no. 5, pp. 547–559, 1969.

[67] J. E. Handschin, "Monte carlo techniques for prediction and filtering of non-linear stochastic processes," *Automatica*, vol. 6, no. 4, pp. 555–563, Jul. 1970.

[68] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Trans. Signal Process.*, vol. 50, no. 3, pp. 736–746, Mar. 2002.

[69] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," in *The Oxford Handbook of Nonlinear Filtering*. Oxford, UK: Oxford University Press, 2009, ch. 24, pp. 656–704.

[70] P. Bickel, B. Li, and T. Bengtsson, "Sharp failure rates for the bootstrap particle filter in high dimensions," in *Pushing the limits of contemporary statist.: Contributions in honor of Jayanta K. Ghosh*. Institute of Mathematical Statist., May 2008, pp. 318–329.

[71] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*, ser. Monographs on Statist. and Applied Probab. Boca Raton, Florida, USA: Chapman & Hall/CRC, 1993, no. 57.

[72] G. Kitagawa, "Monte-Carlo filter and smoother for non-Gaussian nonlinear state space models," *J. Comput. and Graphical Statist.*, vol. 5, no. 1, pp. 1–25, Dec. 1996.

[73] D. Whitley, "A genetic algorithm tutorial," *Statist. and Comput.*, vol. 4, no. 2, pp. 65–85, Jun. 1994.

[74] J. Carpenter, P. Clifford, and P. Fearnhead, "An improved particle filter for non-linear problems," *Proc. IEE Radar Sonar and Navigation*, vol. 146, no. 1, pp. 2–7, Feb. 1999.

[75] H. Sangjin, P. Djuric, and M. Bolic, "Resampling algorithms for particle filters: A computational complexity perspective," *EURASIP J. Adv. Signal Process.*, vol. 15, p. 2267–2277, Nov. 2004.

[76] R. Douc, O. Cappé, and E. Moulines, "Comparison of resampling schemes for particle filtering," in *Proc. IEEE Int. Symposium on Image and Signal Process. and Analysis*, Zagreb, Croatia, Jul. 2005, pp. 64–69.

[77] J. D. Hol, T. B. Schön, and F. Gustafsson, "On resampling algorithms for particle filters," in *Proc. IEEE Nonlinear Statist. Signal Process. Workshop*, Cambridge, UK, Sep. 2006, pp. 79–82.

[78] T. Li, M. Bolic, and P. M. Djuric, "Resampling methods for particle filtering: classification, implementation, and strategies," *IEEE Signal Process. Magazine*, vol. 32, no. 3, pp. 70–86, May 2015.

[79] N. Chopin, "Central limit theorem for sequential Monte Carlo methods and its application to Bayesian inference," *Ann. Statist.*, vol. 32, no. 6, pp. 2385–2411, Dec. 2004.

[80] A. Kong, "A note on importance sampling using standardized weights," Univ. Chicago, Dept. Statist., USA, Tech. Rep. 348, Jul. 1992.

[81] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and Bayesian missing data problems," *J. Amer. Statist. Assoc.*, vol. 89, no. 425, pp. 278–288, Mar. 1994.

[82] T. Bengtsson, P. Bickel, and B. Li, "Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems," in *Probab. Statist.: Essays in Honor of David A. Freedman*. Beachwood, OH, USA: Institute of Mathematical Statist., Apr. 2008, vol. 2, pp. 316–334.

[83] C. Snyder, T. Bengtsson, P. Bickel, and J. Anderson, "Obstacles to high-dimensional particle filtering," *Mon. Weather Rev.*, vol. 136, no. 12, pp. 4629–4640, Dec. 2008.

[84] A. Doucet, S. Godsill, and C. Andrieu, "On sequential Monte Carlo sampling methods for Bayesian filtering," *Stat. Comput.*, vol. 10, no. 3, pp. 197–208, Jul. 2000.

[85] J. Cornebise, É. Moulines, and J. Olsson, "Adaptive methods for sequential importance sampling with application to state space models," *Statist. Comput.*, vol. 18, no. 4, pp. 461–480, Aug. 2008.

[86] S. S. Gu, Z. Ghahramani, and R. E. Turner, "Neural adaptive sequential Monte Carlo," in *Proc. Adv. Neural Info. Process. Syst.*, vol. 28, Montreal, Canada, Dec. 2015, pp. 2629–2637.

[87] M. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *J. Amer. Statist. Assoc.*, vol. 94, no. 446, pp. 590–599, Jun. 1999.

[88] R. Van Der Merwe, A. Doucet, N. De Freitas, and E. Wan, "The unscented particle filter," in *Proc. Adv. Neural Info. Process. Syst.*, Denver, CO, USA, Dec. 2000, pp. 584–590.

[89] A. Doucet, N. d. Freitas, K. P. Murphy, and S. J. Russell, "Rao-Blackwellised particle filtering for dynamic Bayesian networks," in *Proc. Conf. Uncertainty in Artificial Intell.*, San Francisco, CA, USA, Jul. 2000, pp. 176–183.

[90] M. Klaas, N. de Freitas, and A. Doucet, "Toward practical n^2 Monte Carlo: the

marginal particle filter," in *Proc. Conf. Uncertainty in Artificial Intell.*, Edinburgh, Scotland, Jul. 2005, pp. 308–315.

[91] P. J. van Leeuwen, "Nonlinear data assimilation for high-dimensional systems," in *Nonlinear Data Assimilation*. Switzerland: Springer, 2015, ch. 1, pp. 1–73.

[92] M. Ades and P. J. van Leeuwen, "The equivalent-weights particle filter in a high-dimensional system," *Q. J. Royal Met. Soc.*, vol. 141, no. 1, pp. 484–503, Jan. 2015.

[93] A. Beskos, D. Crisan, A. Jasra, K. Kamatani, and Y. Zhou, "A stable particle filter for a class of high-dimensional state-space models," *Adv. Appl. Probab.*, vol. 49, no. 1, p. 24–48, Mar. 2017.

[94] W. R. Gilks and C. Berzuini, "Following a moving target–Monte Carlo inference for dynamic Bayesian models," *J. R. Statist. Soc. B*, vol. 63, no. 1, pp. 127–146, Jan. 2002.

[95] K. Kang, V. Maroulas, and I. D. Schizas, "Drift homotopy particle filter for non-gaussian multi-target tracking," in *Proc. Int. Conf. Info. Fusion*, Salamanca, Spain, Jul. 2014, pp. 1–7.

[96] V. Maroulas and P. Stinis, "Improved particle filters for multi-target tracking," *J. Comput. Phys.*, vol. 231, no. 2, pp. 602–611, Jan. 2012.

[97] V. Maroulas, K. Kang, I. D. Schizas, and M. W. Berry, "A learning drift homotopy particle filter," in *Proc. Int. Conf. Info. Fusion*, Washington, WA, USA, Jul. 2015, pp. 1930–1937.

[98] U. D. Hanebeck, K. Briechle, and A. Rauh, "Progressive Bayes: a new framework for nonlinear state estimation," in *Proc. SPIE Multisensor, Multisource Info. Fusion: Architectures, Algorithms, and Applications*, vol. 5099, Orlando, FL, USA, Apr. 2003, pp. 256–267.

[99] F. Daum and J. Huang, "Particle flow for nonlinear filters with log-homotopy," in *Proc. SPIE Signal and Data Process. Small Targets*, Orlando, FL, USA, Apr. 2008, p. 696918.

[100] F. Daum, J. Huang, A. Noushin, and M. Krichman, "Gradient estimation for particle flow induced by log-homotopy for nonlinear filters," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Orlando, FL, USA, Apr. 2009, p. 733602.

[101] F. Daum and J. Huang, "Exact particle flow for nonlinear filters: seventeen dubious

solutions to a first order linear underdetermined PDE," in *Proc. Asilomar Conf. Signals, Syst. and Comput.*, Pacific Grove, CA, USA, Nov. 2010, pp. 64–71.

[102] F. Daum, J. Huang, and A. Noushin, "Coulomb's law particle flow for nonlinear filters," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, San Diego, CA, USA, Sep. 2011, p. 81370B.

[103] S. Choi, P. Willett, F. Daum, and J. Huang, "Discussion and application of the homotopy filter," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Orlando, FL, USA, May 2011, p. 805021.

[104] F. Daum and J. Huang, "Small curvature particle flow for nonlinear filters," in *Proc. SPIE Signal and Data Process. of Small Targets*, Baltimore, MD, USA, May 2012, p. 83930A.

[105] T. Ding and M. J. Coates, "Implementation of the Daum-Huang exact-flow particle filter," in *Proc. IEEE Statist. Signal Process. Workshop*, Ann Arbor, MI, USA, Aug. 2012, pp. 257–260.

[106] F. Daum and J. Huang, "Particle flow with non-zero diffusion for nonlinear filters," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Baltimore, MD, USA, May 2013, p. 87450P.

[107] ——, "Renormalization group flow and other ideas inspired by physics for nonlinear filters, Bayesian decisions, and transport," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Baltimore, MD, USA, May 2014, p. 90910I.

[108] ——, "Seven dubious methods to mitigate stiffness in particle flow with non-zero diffusion for nonlinear filters, Bayesian decisions, and transport," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Baltimore, MD, USA, May 2014, p. 90920C.

[109] M. A. Khan and M. Ulmke, "Non-linear and non-Gaussian state estimation using log-homotopy based particle flow filters," in *Proc. Sensor Data Fusion: Trends, Solutions, Applications*, Bonn, Germany, Oct. 2014, pp. 1–6.

[110] ——, "Improvements in the implementation of log-homotopy based particle flow filters," in *Proc. Int. Conf. Info. Fusion*, Washington, WA, USA, Jul. 2015, pp. 74–81.

[111] S. Mori, F. Daum, and J. Douglas, "Adaptive step size approach to homotopy-

based particle filtering Bayesian update," in *Proc. Int. Conf. Info. Fusion*, Heidelbreg, Germany, Jul. 2016, pp. 2035–2042.

[112] F. Daum, J. Huang, and A. Noushin, "Generalized Gromov method for stochastic particle flow filters," in *Proc. SPIE Conf. Signal Process., Sensor Fusion, Target Recog.*, Anaheim, CA, USA, May 2017, p. 102000I.

[113] C. Kreucher and K. Bell, "A geodesic flow particle filter for non-thresholded measurements," in *Proc. IEEE Radar Conf.*, Seattle, WA, USA, May 2017, pp. 0891–0896.

[114] F. Daum, J. Huang, and A. Noushin, "New theory and numerical results for Gromov's method for stochastic particle flow filters," in *Proc. Int. Conf. Info. Fusion*, Cambridge, UK, Jul. 2018, pp. 108–115.

[115] K. C. Ward and K. J. DeMars, "Information-based particle flow for high uncertainty estimation," in *Proc. AIAA Scitech Forum*, Orlando, FL, USA, Jan. 2020.

[116] L. Dai and F. Daum, "A new parameterized family of stochastic particle flow filters," *ArXiv e-prints: arXiv 2103.09676*, Mar. 2021.

[117] ——, "Stiffness mitigation in stochastic particle flow filters," *ArXiv e-prints: arXiv 2107.04672*, Jul. 2021.

[118] ——, "Stability and convergence of stochastic particle flow filters," *ArXiv e-prints: arXiv 2108.05255*, Aug. 2021.

[119] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Proc. SPIE Conf. Signal Proc., Sensor Fusion, Target Recog.*, Orlando, FL, USA, Apr. 1997, pp. 182 – 193.

[120] C. Villani, *Optimal Transport: Old and New*. Berlin, Germany: Springer, 2008.

[121] G. Evensen, "The ensemble Kalman filter: Theoretical formulation and practical implementation," *Ocean Dynamics*, vol. 53, no. 4, pp. 343–367, Nov. 2003.

[122] Z. Khan, T. Balch, and F. Dellaert, "MCMC-based particle filtering for tracking a variable number of interacting targets," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 27, no. 11, pp. 1805–1819, Nov. 2005.

[123] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning," *Machine Learning*, vol. 50, no. 1, pp. 5–43, Jan. 2003.

[124] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Phys. Lett. B*, vol. 195, no. 2, pp. 216 – 222, Sep. 1987.

[125] R. M. Neal, "MCMC using Hamiltonian dynamics," in *Handbook of Markov Chain Monte Carlo*. Boca Raton, USA: Chapman and Hall/CRC, May 2011, ch. 5, pp. 113–162.

[126] G. O. Roberts and O. Stramer, "Langevin diffusions and Metropolis-Hastings algorithms," *Method. Comput. Appl. Probab.*, vol. 4, pp. 337–357, Dec. 2002.

[127] A. Brockwell, P. D. Moral, and A. Doucet, "Sequentially interacting Markov chain Monte Carlo methods," *Ann. Statist.*, vol. 38, no. 6, pp. 3387–3411, Nov. 2010.

[128] P. J. Rossky, J. D. Doll, and H. L. Friedman, "Brownian dynamics as smart Monte Carlo simulation," *J. Chem. Phys.*, vol. 69, pp. 4628–4633, Jun. 1978.

[129] M. Girolami and B. Calderhead, "Riemann manifold Langevin and Hamiltonian Monte Carlo methods," *J. R. Stat. Soc. B*, vol. 73, no. 2, pp. 123–214, Mar. 2011.

[130] Y. Lecun and Y. Bengio, *Convolutional Networks for Images, Speech and Time Series*. Cambridge, MA, USA: MIT Press, Jan. 1995, pp. 255–258.

[131] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Info. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 2012, p. 1097–1105.

[132] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sciences, USA*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[133] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comp.*, vol. 9, p. 1735–1780, Nov. 1997.

[134] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. Int. Conf. Learning Representations*, San Juan, Puerto Rico, May 2016.

[135] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. Zemel, "Graph partition neural networks for semi-supervised classification," *ArXiv e-prints: arXiv 1803.06272*, Mar. 2018.

[136] W. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large

graphs," in *Proc. Adv. Neural Info. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 1024–1034.

[137] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learning Representations*, Banff, Canada, Apr. 2014.

[138] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2015, pp. 2224–2232.

[139] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. Int. Conf. Learning Representations*, New Orleans, LA, USA, May 2019.

[140] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *ArXiv e-prints: arXiv 1806.01261*, 2018.

[141] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[142] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Apr. 2021.

[143] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *ArXiv e-prints: arXiv 1506.05163*, Jun. 2015.

[144] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Magazine*, vol. 30, no. 3, pp. 83–98, May 2013.

[145] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph

convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 997–109, Jan. 2019.

[146] A. Wijesinghe and Q. Wang, "DFNets: Spectral CNNs for graphs with feedback-looped filters," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2019, pp. 6009–6020.

[147] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. Adv. Neural Info. Process. Syst.*, Barcelona, Spain, Dec. 2016.

[148] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. IEEE Conf. Comp. Vision and Pattern Recog.*, Honolulu, US, Jul. 2017.

[149] X. Bresson and T. Laurent, "Residual gated graph ConvNets," *ArXiv e-prints: arXiv 1711.07553*, Nov. 2017.

[150] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. Adv. Neural Info. Process. Syst.*, Barcelona, Spain, Dec. 2016, pp. 2244–2252.

[151] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comp. Vision and Pattern Recog.*, Honolulu, HI, USA, Jul. 2017.

[152] R. Anirudh and J. J. Thiagarajan, "Bootstrapping graph convolutional neural networks for autism spectrum disorder classification," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, Brighton, UK, May 2019, pp. 3197–3201.

[153] F. Petroski Such, S. Sah, M. Dominguez, S. Pillai, C. Zhang, A. Michael, N. Cahill, and R. Ptucha, "Robust spatial filtering with graph convolutional neural networks," *IEEE J. Sel. Topics Signal Process.*, vol. 11, no. 6, pp. 884–896, Sep. 2017.

[154] F. Monti, O. Shchur, A. Bojchevski, O. Litany, S. Günnemann, and M. M. Bronstein, "Dual-primal graph convolutional networks," *ArXiv e-prints: arXiv 1806.00770*, Jun. 2018.

[155] J. Chen, T. Ma, and C. Xiao, "FastGCN: fast learning with graph convolutional networks via importance sampling," in *Proc. Int. Conf. Learning Representations*, Vancouver, Canada, May 2018.

[156] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. Int. Conf. Machine Learning*, Stockholm, Sweden, Jul. 2018.

[157] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2018, p. 4563–4572.

[158] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," in *Proc. Adv. Neural Info. Process. Syst.*, Dec. 2019.

[159] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in *Proc. Int. Conf. Learning Representations*, Virtual, May 2020.

[160] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. & Data Mining*, Aug. 2019.

[161] H. NT and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *ArXiv e-prints: arXiv 1905.09550*, May 2019.

[162] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. Int. Conf. Machine Learning*, Stockholm, Sweden, Jul. 2018, pp. 5449–5458.

[163] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. AAAI Conf. Artificial Intelligence*, S. A. McIlraith and K. Q. Weinberger, Eds., New Orleans, LA, USA, Feb. 2018, pp. 3538–3545.

[164] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proc. AAAI Conf. Artificial Intelligence*, New York, NY, USA, Feb. 2020.

[165] L.-P. Xhonneux, M. Qu, and J. Tang, "Continuous graph neural networks," in *Proc. Int. Conf. Machine Learning*, Virtual, Jul. 2020, pp. 10 432–10 441.

[166] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. Int. Conf. Learning Representations*,

New Orleans, LA, USA, May 2019.

[167] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. & Data Mining*, virtual, Aug. 2020, p. 338–348.

[168] Y. Rong, W. Huang, T. Xu, and J. Huang, "DropEdge: Towards deep graph convolutional networks on node classification," in *Proc. Int. Conf. Learning Representations*, Virtual, May 2020.

[169] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian, "Bayesian graph neural networks with adaptive connection sampling," in *Proc. Int. Conf. Machine Learning*, Virtual, Jul. 2020, pp. 4094–4104.

[170] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.

[171] P. Ravikumar, M. J. Wainwright, G. Raskutti, and B. Yu, "High-dimensional covariance estimation by minimizing $\ell 1$-penalized log-determinant divergence," *Electron. J. Statist.*, vol. 5, pp. 935–980, Nov. 2008.

[172] V. Kalofolias, "How to learn a graph from smooth signals," in *Proc. Int. Conf. Artificial Intell. and Statist.*, Cadiz, Spain, May 2016, pp. 920–929.

[173] E. Kolaczyk, *Statistical Analysis of Network Data: Methods and Models*. New York, NY, USA: Springer, 2009.

[174] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proc. Int. Conf. World Wide Web*, Hyderabad, India, Mar. 2011, p. 577–586.

[175] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.

[176] S. Hassan-Moghaddam, N. K. Dhingra, and M. R. Jovanović, "Topology identification of undirected consensus networks via sparse inverse covariance estimation," in *IEEE Conf. Decision and Control*, Las Vegas, NV, USA, Dec. 2016, pp. 4624–4629.

[177] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under laplacian and structural constraints," *IEEE J. Sel. Topics in Signal Process.*, vol. 11, no. 6, pp.

825–841, Sep. 2017.

[178] C. Hu, L. Cheng, J. Sepulcre, G. El Fakhri, Y. M. Lu, and Q. Li, "A graph theoretical regression model for brain connectivity learning of Alzheimer's disease," in *IEEE Int. Symp. Biomed. Imaging*, San Francisco, CA, USA, Apr. 2013, pp. 616–619.

[179] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning laplacian matrix in smooth graph signal representations," *IEEE Trans. Sig. Process.*, vol. 64, no. 23, pp. 6160–6173, Dec. 2016.

[180] V. Kalofolias and N. Perraudin, "Large scale graph learning from smooth signals," in *Proc. Int. Conf. Learning Representations*, New Orleans, LA, USA, May 2019.

[181] D. Thanou, X. Dong, D. Kressner, and P. Frossard, "Learning heat diffusion graphs," *IEEE Trans. Signal and Info. Process. over Networks*, vol. 3, no. 3, pp. 484–499, Sep. 2017.

[182] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Process. Magazine*, vol. 36, pp. 16–43, May 2019.

[183] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2001.

[184] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, Apr. 2018.

[185] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74, no. 1, p. 47–97, Jan. 2002.

[186] P. Erdös and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.

[187] F. Caron and E. B. Fox, "Sparse graphs using exchangeable random measures," *J. Royal Statist. Soc. Ser. B (Statist. Method.)*, vol. 79, no. 5, pp. 1295–1366, Sep. 2017.

[188] B. K. Fosdick, D. B. Larremore, J. Nishimura, and J. Ugander, "Configuring random graph models with fixed degree sequences," *SIAM Review*, vol. 60, no. 2, pp. 315–355, May 2018.

[189] G. Casiraghi, "Analytical formulation of the block-constrained configuration model," *ArXiv e-prints: arXiv 1811.05337*, Nov. 2018.

[190] M. Drobyshevskiy and D. Turdakov, "Random graph modeling: A survey of the concepts," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–36, Dec. 2019.

[191] B. Ball, B. Karrer, and M. E. J. Newman, "Efficient and principled method for detecting communities in networks," *Phys. Rev. E*, vol. 84, no. 3, p. 036103, Sep. 2011.

[192] H. Mahmoud, F. Masulli, S. Rovetta, and G. Russo, "Community detection in protein-protein interaction networks using spectral and graph approaches," in *Proc. Int. Meeting Comput. Intell. Methods for Bioinformatics and Biostatistics*, Jul. 2014, pp. 62–75.

[193] K. He, Y. Li, S. Soundarajan, and J. E. Hopcroft, "Hidden community detection in social networks," *Inform. Sci.*, vol. 425, pp. 92–106, Jan. 2018.

[194] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Natl. Acad. Sciences, USA*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006.

[195] K. Nowicki and T. A. B. Snijders, "Estimation and prediction for stochastic blockstructures," *J. Amer. Statist. Assoc.*, vol. 96, no. 455, pp. 1077–1087, Sep. 2001.

[196] B. Karrer and M. E. Newman, "Stochastic blockmodels and community structure in networks," *Physical review E*, vol. 83, no. 1, p. 016107, 2011.

[197] T. P. Peixoto, "Bayesian stochastic blockmodeling," *ArXiv e-prints: arXiv 1705.10225*, Feb. 2017.

[198] E. Abbe, "Community detection and stochastic block models," *Found. and Trends Commun. and Inform. Theory*, vol. 14, no. 1-2, pp. 1–162, Jun. 2018.

[199] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, "Mixed membership stochastic blockmodels," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, Canada, Dec. 2009, pp. 33–40.

[200] P. K. Gopalan, S. Gerrish, M. Freedman, D. Blei, and D. Mimno, "Scalable inference of overlapping communities," in *Proc. Adv. Neural Info. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 2012.

[201] Y. Li, L. Zhao, and M. J. Coates, "Particle flow for particle filtering," in *Proc. IEEE*

Int. Conf. Acoust., Speech and Signal Process., Shanghai, China, Mar. 2016.

[202] P. Latouche, E. Birmelé, and C. Ambroise, "Overlapping stochastic block models with application to the french political blogosphere," *Ann. Applied Statist.*, vol. 5, Oct. 2009.

[203] K. T. Miller, T. L. Griffiths, and M. I. Jordan, "Nonparametric latent feature models for link prediction," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2009, p. 1276–1284.

[204] Y. Zhao, E. Levina, and J. Zhu, "Consistency of community detection in networks under degree-corrected stochastic block models," *Ann. Statist*, vol. 40, no. 4, pp. 2266–2292, Aug. 2012.

[205] C. Gao, Z. Ma, A. Y. Zhang, and H. H. Zhou, "Community detection in degree-corrected block models," *Ann. Statist.*, vol. 46, no. 5, pp. 2153–2185, Oct. 2018.

[206] A. A. Amini, A. Chen, P. J. Bickel, and E. Levina, "Pseudo-likelihood methods for community detection in large sparse networks," *Ann. Statist.*, vol. 41, no. 4, pp. 2097–2122, 2013.

[207] L. Peng and L. Carvalho, "Bayesian degree-corrected stochastic blockmodels for community detection," *Electron. J. Statist.*, vol. 10, no. 2, pp. 2746–2779, Sep. 2016.

[208] S. Pal and M. Coates, "Scalable MCMC in degree corrected stochastic block models," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, Brighton, UK, May 2019, pp. 5461–5465.

[209] K. Bringmann, R. Keusch, and J. Lengler, "Geometric inhomogeneous random graphs," *Theoretical Comput. Science*, vol. 760, pp. 35–54, Feb. 2019.

[210] V. Veitch and D. M. Roy, "The class of random graphs arising from exchangeable random measures," *ArXiv e-prints: arXiv 1512.03099*, Dec. 2015.

[211] F. Regol, S. Pal, J. Sun, Y. Zhang, Y. Geng, and M. Coates, "Node copying: A random graph model for effective graph sampling," *Signal Process.*, vol. 192, p. 108335, Mar. 2022.

[212] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," in *Proc. Int. Joint Conf. Artificial Intell.*, Stockholm, Sweden, Jul. 2018, pp. 2609–2615.

[213] N. Mehta, L. C. Duke, and P. Rai, "Stochastic blockmodels meet graph neural

networks," in *Proc. Int. Conf. Machine Learning*, Long Beach, CA, USA, Jun. 2019, pp. 4466–4474.

[214] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "GraphGAN: Graph representation learning with generative adversarial nets," in *Proc. AAAI Conf. Artificial Intell.*, San Francisco, CA, USA, Feb. 2017, pp. 2508–2515.

[215] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proc. Int. Conf. Machine Learning*, Stockholm, Sweden, Jul. 2018, pp. 609–618.

[216] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *Proc. Int. Conf. Artificial Neural Networks*, Rhodes, Greece, Oct. 2018, pp. 412–422.

[217] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2018, pp. 7113–7124.

[218] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," in *Proc. Workshop Theoretical Foundations and Applications of Deep Generative Models, Int. Conf. Machine Learning*, Stockholm, Sweden, Jul. 2018.

[219] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia, "Learning deep generative models of graphs," *ArXiv e-prints: arXiv 1803.03324*, Mar. 2018.

[220] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. J. Swersky, "Graph normalizing flows," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2019, pp. 13 578–13 588.

[221] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proc. Int. Conf. Machine Learning*, Stockholm, Sweden, Jul. 2018, pp. 5708–5717.

[222] R. Liao, Y. Li, Y. Song, S. Wang, C. Nash, W. L. Hamilton, D. Duvenaud, R. Urtasun, and R. S. Zemel, "Efficient graph generation with graph recurrent attention networks," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2019, pp. 4255–4265.

[223] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal

dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *Proc. Int. Conf. Machine Learning*, Edinburgh, UK, Jul. 2012, p. 1881–1888.

[224] J. Bayer and C. Osendorfer, "Learning stochastic recurrent networks," *ArXiv e-prints: arXiv 1411.7610*, Nov. 2014.

[225] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood, "Auto-encoding sequential Monte Carlo," in *Proc. Int. Conf. Learning Representations*, Vancouver, Canada, May 2018.

[226] C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh, "Filtering variational objectives," in *Proc. Adv. Neural Info. Process. Syst.*, Long Beach, CA, USA, Dec. 2017.

[227] C. Naesseth, S. Linderman, R. Ranganath, and D. Blei, "Variational sequential Monte Carlo," in *Proc. Int. Conf. Artificial Intell. and Statist.*, Lanzarote, Canary Islands, Apr. 2018, pp. 968–977.

[228] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.

[229] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Melbourne, Australia: OTexts, 2018.

[230] J. D. Hamilton, *Time Series Analysis*, 1st ed. Princeton University Press, Jan. 1994.

[231] S. J. Taylor and B. Letham, "Forecasting at scale," *The Amer. Statist.*, vol. 72, no. 1, pp. 37–45, Apr. 2018.

[232] F. R. Bach and M. I. Jordan, "Learning graphical models for stationary time series," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2189–2199, Aug 2004.

[233] J. Songsiri and L. Vandenberghe, "Topology selection in graphical models of autoregressive processes," *J. Machine Learning Research*, vol. 11, p. 2671–2705, Dec. 2010.

[234] A. Bolstad, B. D. Van Veen, and R. Nowak, "Causal network inference via group sparse regularization," *IEEE Trans. Signal Process.*, vol. 59, no. 6, pp. 2628–2641, Jun. 2011.

[235] J. Mei and J. M. F. Moura, "Signal processing on graphs: Causal modeling of unstructured data," *IEEE Trans. Signal Process.*, vol. 65, no. 8, pp. 2077–2092, Apr. 2017.

[236] E. Isufi, A. Loukas, N. Perraudin, and G. Leus, "Forecasting time series with VARMA recursions on graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 18, pp. 4870–4885, Jul. 2019.

[237] N. Lim, F. d'Alché Buc, C. Auliac, and G. Michailidis, "Operator-valued kernel-based vector autoregressive models for network inference," *Machine Learning*, vol. 99, no. 3, pp. 489–513, Dec. 2014.

[238] Y. Shen, G. B. Giannakis, and B. Baingana, "Nonlinear structural vector autoregressive models with application to directed brain networks," *IEEE Trans. Signal Process.*, vol. 67, no. 20, pp. 5325–5339, Oct. 2019.

[239] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PloS One*, vol. 12, no. 7, Jul. 2017.

[240] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *Proc. ACM SIGIR Int. Conf. Research & Development in Info. Retrieval*, Ann Arbor, MI, USA, Jul. 2018, pp. 95–104.

[241] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. W. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," in *Proc. Int. Joint Conf. Artificial Intell.*, Melbourne, Australia, Aug. 2017.

[242] Y.-Y. Chang, F.-Y. Sun, Y.-H. Wu, and S.-D. Lin, "A memory-network based solution for multivariate time-series forecasting," *ArXiv e-prints: arXiv 1809.02105*, Sep. 2018.

[243] T. Guo and T. Lin, "Multi-variable LSTM neural network for autoregressive exogenous model," *ArXiv e-prints: arXiv 1806.06384*, Jun. 2018.

[244] L. Munkhdalai, T. Munkhdalai, K. H. Park, T. Amarbayasgalan, E. Erdenebaatar, H. W. Park, and K. H. Ryu, "An end-to-end adaptive input selection with dynamic weights for forecasting multivariate time series," *IEEE Access*, vol. 7, pp. 99 099–99 114, Jul. 2019.

[245] F. Liu, Y. Lu, and M. Cai, "A hybrid method with adaptive sub-series clustering and attention-based stacked residual LSTMs for multivariate time series forecasting," *IEEE Access*, vol. 8, pp. 62 423–62 438, Mar. 2020.

[246] H.-F. Yu, N. Rao, and I. S. Dhillon, "Temporal regularized matrix factorization for

high-dimensional time series prediction," in *Proc. Adv. Neural Info. Process. Syst.*, Barcelona, Spain, Dec. 2016, p. 847–855.

[247] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[248] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, Jun. 1990.

[249] B. Kosko, "Bidirectional associative memories," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 49–60, Feb. 1988.

[250] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *ArXiv e-prints: arXiv 1406.1078*, Sep. 2014.

[251] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv e-prints: arXiv 1308.0850*, Aug. 2013.

[252] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," *ArXiv e-prints: arXiv 1609.03499*, Sep. 2016.

[253] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "DRAW: A recurrent neural network for image generation," in *Proc. Int. Conf. Machine Learning*, Lille, France, Jul. 2015, pp. 1462–1471.

[254] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *Int. J. Forecasting*, vol. 36, no. 1, pp. 75 – 85, Mar. 2020.

[255] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[256] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Machine Learning*, Atlanta, GA, USA, Jun. 2013, pp. 1310–1318.

[257] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2007, p. 153–160.

[258] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Info. Process. Syst.*, Long Beach, CA, USA, Dec. 2017.

[259] Z. Wu and N. E. Huang, "Ensemble empirical mode decomposition: A noise-assisted data analysis method," *Adv. Adaptive Data Analysis*, vol. 01, no. 01, pp. 1–41, 2009.

[260] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Proc. Kernel workshop, Annual Conf. Learning Theory*, Washington, WA, USA, Aug. 2003, pp. 144–158.

[261] Z. Chen and A. Cichocki, "Nonnegative matrix factorization with temporal smoothness and/or spatial decorrelation constraints," Laboratory for Advanced Brain Signal Processing, RIKEN Brain Science Institute, Wako-shi, Japan, Tech. Rep. 68, 2005.

[262] M. Roughan, Y. Zhang, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices (extended version)," *IEEE/ACM Trans. Networking*, vol. 20, no. 3, pp. 662–676, Jun. 2012.

[263] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. Int. Conf. Learning Representations*, San Juan, Puerto Rico, May 2016.

[264] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka, "A multi-horizon quantile recurrent forecaster," *ArXiv e-prints: arXiv 1711.11053*, Jun. 2017.

[265] D. Salinas, M. Bohlke-Schneider, L. Callot, R. Medico, and J. Gasthaus, "High-dimensional multivariate forecasting with low-rank Gaussian copula processes," in *Proc. Adv. Neural Info. Process. Syst.*, Vancouver, Canada, Dec. 2019.

[266] R. Kurle, S. S. Rangapuram, E. de Bézenac, S. Günnemann, and J. Gasthaus, "Deep Rao-Blackwellised particle filters for time series forecasting," in *Proc. Adv. Neural Info. Process. Syst.*, Virtual, Dec. 2020.

[267] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 43, no. 11, pp. 3964–3979, Nov. 2020.

[268] E. de Bézenac, S. S. Rangapuram, K. Benidis, M. Bohlke-Schneider, R. Kurle, L. Stella, H. Hasson, P. Gallinari, and T. Januschowski, "Normalizing Kalman filters for multivariate time series analysis," in *Proc. Adv. Neural Info. Process. Syst.*, Virtual,

Dec. 2020.

[269] K. Rasul, A.-S. Sheikh, I. Schuster, U. Bergmann, and R. Vollgraf, "Multivariate probabilistic time series forecasting via conditioned normalizing flows," in *Proc. Int. Conf. Learning Representations*, Virtual, May 2021.

[270] L. Li, J. Yan, X. Yang, and Y. Jin, "Learning interpretable deep state space model for probabilistic time series forecasting," in *Proc. Int. Joint Conf. Artificial Intell.*, Macao, China, Aug. 2019, pp. 2901–2908.

[271] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," in *Proc. Adv. Neural Info. Process. Syst.*, Barcelona, Spain, Dec. 2016, p. 3745–3753.

[272] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *Proc. Int. Conf. Learning Representations*, Toulon, France, Apr. 2017.

[273] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Proc. Adv. Neural Info. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, p. 2335–2344.

[274] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma, "VideoFlow: A conditional flow-based model for stochastic video generation," in *Proc. Int. Conf. Learning Representations*, Virtual, May 2020.

[275] D. Gammelli and F. Rodrigues, "Recurrent flow networks: A recurrent latent variable model for spatio-temporal density modelling," *ArXiv e-prints: arXiv 2006.05256*, Jun. 2020.

[276] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. Int. Joint Conf. Artificial Intell.*, Stockholm, Sweden, Jul. 2018.

[277] Q. Zhang, J. Chang, G. Meng, S. Xiang, and C. Pan, "Spatio-temporal graph structure learning for traffic forecasting," in *Proc. AAAI Conf. Artificial Intell.*, New York, NY, USA, Feb. 2020, pp. 1177–1185.

[278] B. N. Oreshkin, A. Amini, L. Coyle, and M. J. Coates, "FC-GAGA: Fully Connected Gated Graph Architecture for spatio-temporal traffic forecasting," in *Proc. AAAI Conf. Artificial Intell.*, Virtual, Feb. 2021.

[279] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proc. Int. Conf. Machine Learning*, Sydney, Australia, Aug. 2017, p. 1243–1252.

[280] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction," *ArXiv e-prints: arXiv 1811.05320*, Nov. 2018.

[281] R. Zhao, K. Wang, H. Su, and Q. Ji, "Bayesian graph convolution LSTM for skeleton based action recognition," in *Proc. IEEE/CVF Int. Conf. Comp. Vision*, Seoul, Korea, Oct. 2019.

[282] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," *IEEE Trans. Intell. Transport. Syst.*, vol. 21, no. 11, pp. 4883–4894, Nov. 2020.

[283] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2015, p. 1171–1179.

[284] Y. Huang, Y. Weng, S. Yu, and X. Chen, "Diffusion convolutional recurrent neural network with rank influence learning for traffic forecasting," in *Proc. IEEE Int. Conf. Big Data Science and Engineering*, Rotorua, New Zealand, Aug 2019, pp. 678–685.

[285] C. Chen, K. Li, S. G. Teo, X. Zou, K. Wang, J. Wang, and Z. Zeng, "Gated residual recurrent graph neural networks for traffic prediction," in *Proc. AAAI Conf. Artificial Intell.*, Honolulu, HI, USA, Feb. 2019, pp. 485–492.

[286] R. Huang, C. Huang, Y. Liu, G. Dai, and W. Kong, "LSGCN: Long short-term traffic prediction with graph convolutional networks," in *Proc. Int. Joint Conf. Artificial Intell.*, Virtual, Jan. 2021, pp. 2355–2361.

[287] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for deep spatial-temporal graph modeling," in *Proc. Int. Joint Conf. Artificial Intell.*, Macao, China, Aug. 2019, pp. 1907–1913.

[288] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. ACM*

SIGKDD Int. Conf. Knowl. Disc. Data Mining, Virtual, Aug. 2020, pp. 753–763.

[289] B. Yu, M. Li, J. Zhang, and Z. Zhu, "3D graph convolutional networks with temporal graphs: A spatial information free framework for traffic forecasting," *ArXiv e-prints: arXiv 1903.00919*, Mar. 2019.

[290] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, Seattle, WA, USA, Jul. 1994, p. 359–370.

[291] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He, "Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting," in *Proc. AAAI Conf. Artificial Intell.*, Honolulu, HI, USA, Feb. 2019, pp. 890–897.

[292] C. Song, Y. Lin, S. Guo, and H. Wan, "Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting," in *Proc. AAAI Conf. Artificial Intell.*, New York, NY, USA, Feb. 2020, pp. 914–921.

[293] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conf. Artificial Intell.*, Honolulu, HI, USA, Feb. 2019.

[294] L. Bai, L. Yao, S. Kanhere, X. Wang, and Q. Sheng, "STG2Seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting," in *Proc. Int. Joint Conf. Artificial Intell.*, Macao, China, Aug. 2019, pp. 1981–1987.

[295] C. Zheng, X. Fan, C. Wang, and J. Qi, "GMAN: A Graph Multi-Attention Network for Traffic Prediction," in *Proc. AAAI Conf. Artificial Intell.*, New York, NY, USA, Feb. 2020.

[296] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, San Francisco, CA, USA, Aug. 2016, p. 855–864.

[297] X. Shi, H. Qi, Y. Shen, G. Wu, and B. Yin, "A spatial-temporal attention approach for traffic prediction," *IEEE Trans. Intell. Transport. Syst.*, pp. 1–10, Apr. 2020.

[298] C. Park, C. Lee, H. Bahng, T. won, K. Kim, S. Jin, S. Ko, and J. Choo, "ST-GRAT: A spatio-temporal graph attention network for traffic forecasting," in *Proc. AAAI Conf.*

Artificial Intell., New York, NY, USA, Feb. 2020.

[299] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proc. Int. Conf. World Wide Web*, Florence, Italy, May 2015, pp. 1067–1077.

[300] M. Xu, W. Dai, C. Liu, X. Gao, W. Lin, G.-J. Qi, and H. Xiong, "Spatial-temporal transformer networks for traffic flow forecasting," *ArXiv e-prints: arXiv 2001.02908*, Jan. 2020.

[301] B. Yu, H. Yin, and Z. Zhu, "ST-UNet: A spatio-temporal U-network for graph-structured time series modeling," *ArXiv e-prints: arXiv 1903.05631*, Mar. 2019.

[302] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. Medical Image Computing and Computer-Assisted Intervention*, Munich, Germany, Oct. 2015, pp. 234–241.

[303] G. E. Hinton and T. J. Sejnowski, "Optimal perceptual inference," in *Proc. IEEE Conf. Comp. Vision and Pattern Recog.*, Washington, WA, USA, Jun. 1983.

[304] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2015, pp. 2980–2988.

[305] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learning Representations*, Banff, Canada, Apr. 2014.

[306] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther, "Sequential neural models with stochastic layers," *ArXiv e-prints: arXiv 1605.07571*, May 2016.

[307] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Proc. Adv. Neural Info. Process. Syst.*, Long Beach, CA, USA, Dec. 2017.

[308] M. Karl, M. Soelch, J. Bayer, and P. Van der Smagt, "Deep variational Bayes filters: Unsupervised learning of state space models from raw data," in *Proc. Int. Conf. Learning Representations*, Toulon, France, Apr. 2017.

[309] C. L. C. Mattos, Z. Dai, A. Damianou, J. Forth, G. A. Barreto, and N. D. Lawrence, "Recurrent Gaussian processes," in *Proc. Int. Conf. Learning Representations*, San Juan, Puerto Rico, May 2016.

[310] A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe, "Probabilistic recurrent state-space models," *ArXiv e-prints: arXiv 1801.10395*, Jan. 2018.

[311] T. B. Schön, A. Wills, and B. Ninness, "System identification of nonlinear state-space models," *Automatica*, vol. 47, no. 1, pp. 39–49, Jan. 2011.

[312] G. Poyiadjis, A. Doucet, and S. S. Singh, "Particle approximations of the score and observed information matrix in state space models with application to parameter estimation," *Biometrika*, vol. 98, no. 1, pp. 65–80, Feb. 2011.

[313] C. Andrieu, A. Doucet, and R. Holenstein, "Particle Markov Chain Monte Carlo methods," *J. Royal Statist. Soc. Ser. B (Statist. Method.)*, vol. 72, no. 3, pp. 269–342, May 2010.

[314] C. M. Bishop, "Mixture density networks," Dept. Comp. Science and Appl. Math., Aston Univ., UK, Tech. Rep. 004, Feb. 1994.

[315] Y. Burda, R. Grosse, and R. Salakhutdinov, "Importance weighted autoencoders," in *Proc. Int. Conf. Learning Representations*, San Juan, Puerto Rico, May 2016.

[316] X. Zheng, M. Zaheer, A. Ahmed, Y. Wang, E. P. Xing, and A. J. Smola, "State space LSTM models with particle MCMC inference," *ArXiv e-prints: arxiv 1711.11179*, Nov. 2017.

[317] P. Karkus, D. Hsu, and W. S. Lee, "Particle filter networks with application to visual localization," in *Proc. Conf. Robot Learning*, Zurich, Switzerland, Oct. 2018, pp. 169–178.

[318] X. Ma, P. Karkus, D. Hsu, and W. S. Lee, "Particle filter recurrent neural networks," in *Proc. AAAI Conf. Artificial Intell.*, New York, NY, USA, Feb. 2020, pp. 5101–5108.

[319] J. S. Denker and Y. Lecun, "Transforming neural-net output levels to probability distributions," in *Proc. Adv. Neural Inf. Process. Syst.*, Denver, CO, USA, Dec. 1991.

[320] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of Bayesian neural networks," in *Proc. Int. Conf. Machine Learning*, Lille, France, Jul. 2015, pp. 1861–1869.

[321] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. Int. Conf. Machine Learning*, New York,

NY, USA, Jun. 2016, pp. 1050–1059.

[322] S. Sun, C. Chen, and L. Carin, "Learning structured weight uncertainty in Bayesian neural networks," in *Proc. Int. Conf. Artificial Intell. and Statist.*, Ft. Lauderdale, FL, USA, Apr. 2017, pp. 1283–1292.

[323] C. Louizos and M. Welling, "Multiplicative normalizing flows for variational Bayesian neural networks," in *Proc. Int. Conf. Machine Learning*, Sydney, Australia, Aug. 2017, pp. 2218–2227.

[324] A. Korattikara, V. Rathod, K. Murphy, and M. Welling, "Bayesian dark knowledge," in *Proc. Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2015.

[325] C. Li, C. Chen, D. Carlson, and L. Carin, "Pre-conditioned stochastic gradient Langevin dynamics for deep neural networks," in *Proc. AAAI Conf. Artificial Intell.*, Phoenix, ARI, USA, Feb. 2016, p. 1788–1794.

[326] W. Li, S. Ahn, and M. Welling, "Scalable MCMC for mixed membership stochastic blockmodels," in *Proc. Int. Conf. Artificial Intell. and Statist.*, Cadiz, Spain, May 2016, pp. 723–731.

[327] D. J. C. MacKay, *Maximum entropy and Bayesian methods*. Springer Netherlands, 1996, ch. Hyperparameters: Optimize, or Integrate Out?, pp. 43–59.

[328] S. Patterson and Y. W. Teh, "Stochastic gradient Riemannian Langevin dynamics on the probability simplex," in *Proc. Adv. Neural Info. Process. Syst.*, Lake Tahoe, NV, USA, Dec. 2013.

[329] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, p. 93, Sep. 2008.

[330] G. Namata, B. London, L. Getoor, and B. Huang, "Query-driven active surveying for collective classification," in *Proc. Workshop on Mining and Learning with Graphs, Int. Conf. Machine Learning*, Edinburgh, Scotland, Jun. 2012.

[331] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. Int Conf. Machine Learning*, New York, NY, USA, Jun. 2016, p. 40–48.

[332] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learning Representations*, San Diego, CA, USA, May

2015.

[333] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, London, UK, Aug. 2018.

[334] L. Tang and H. Liu, "Leveraging social media networks for classification," *Data Mining and Knowl. Disc.*, pp. 447–478, Jan. 2011.

[335] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, New York, New York, USA, Aug. 2014, pp. 701–710.

[336] T. Gneiting and A. E. Raftery, "Strictly proper scoring rules, prediction, and estimation," *J. Amer. Statist. Assoc.*, vol. 102, no. 477, pp. 359–378, Mar. 2007.

[337] C. Chen, K. Petty, and A. Skabardonis, "Freeway performance measurement system: Mining loop detector data," *Transport. Research Record*, vol. 1748, Jan. 2000.

[338] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[339] W. Chen, L. Chen, Y. Xie, W. Cao, Y. Gao, and X. Feng, "Multi-range attentive bicomponent graph convolutional network for traffic forecasting," in *Proc. AAAI Conf. Artificial Intell.*, New York, NY, USA, Feb. 2020, pp. 3529–3536.

[340] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang, "Urban traffic prediction from spatio-temporal data using deep meta learning," in *Proc. ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, Anchorage, AK, USA, Aug. 2019, p. 1720–1730.

[341] W. Chun-Hsin, H. Jan-Ming, and L. D. T., "Travel-time prediction with support vector regression," *IEEE Trans. Intell. Transport. Syst.*, vol. 5, no. 4, pp. 276–281, Dec. 2004.

[342] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Info. Process. Syst.*, Dec. 2014, pp. 3104–3112.

[343] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. Workshop on Deep Learning, Adv. Neural Info. Process. Syst.*, Montreal, Canada, Dec. 2014.

[344] A. Finke, A. Doucet, and A. M. Johansen, "Limit theorems for sequential MCMC methods," *Adv. Applied Probab.*, vol. 52, no. 2, p. 377–403, Jul. 2020.

[345] C. Fantacci, B. Vo, B. Vo, G. Battistelli, and L. Chisci, "Robust fusion for multisensor multiobject tracking," *IEEE Signal Process. Lett.*, vol. 25, no. 5, pp. 640–644, May 2018.

[346] T. Brehard and V. Krishnamurthy, "Optimal data incest removal in Bayesian decentralized estimation over a sensor network," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process.*, vol. 3, Honolulu, Hawaii, USA, Apr. 2007, pp. III–173–III–176.

[347] J. Liu, I. Nevat, P. Zhang, and G. W. Peters, "Multimodal data fusion in sensor networks via copula processes," in *Proc. IEEE Wireless Comm. Networking Conf.*, San Francisco, CA, USA, Mar. 2017, pp. 1–6.

[348] S. Pal and M. Coates, "Particle flow particle filter using Gromov's method," in *Proc. IEEE Int. Workshop Comput. Adv. Multi-Sensor Adaptive Process.*, Guadeloupe, West Indies, Dec. 2019.

[349] M. Ottobre, "Markov Chain Monte Carlo and irreversibility," *Reports on Math. Physics*, vol. 77, no. 3, pp. 267–292, Jun. 2016.

[350] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet, "The bouncy particle sampler: A non-reversible rejection-free Markov chain Monte Carlo method," *J. Amer. Statist. Assoc.*, vol. 113, no. 522, pp. 855–867, Jun. 2018.

[351] C. Sherlock and A. H. Thiery, "A discrete bouncy particle sampler," *ArXiv e-prints: arXiv 1707.05200*, Jul. 2017.

[352] P. Fearnhead, J. Bierkens, M. Pollock, and G. O. Roberts, "Piecewise deterministic Markov processes for continuous-time Monte Carlo," *Statist. Science*, vol. 33, no. 3, pp. 386–412, Aug. 2018.

[353] J. Bierkens, P. Fearnhead, and G. Roberts, "The zig-zag process and super-efficient sampling for Bayesian analysis of big data," *Ann. Statist.*, vol. 47, no. 3, pp. 1288–1320, Jun. 2019.

[354] S. Pal and M. Coates, "Sequential MCMC with the discrete bouncy particle sampler," in *Proc. IEEE Statist. Signal Process. Workshop*, Freiburg, Germany, Jun. 2018, pp. 663–667.

[355] X. Cheng, N. S. Chatterji, P. L. Bartlett, and M. I. Jordan, "Underdamped Langevin

MCMC: A non-asymptotic analysis," in *Proc. Conf. Learning Theory*, vol. 75, Jul. 2018, pp. 300–323.

[356] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-GCN: Geometric graph convolutional networks," in *Proc. Int. Conf. Learning Representations*, Virtual, Apr. 2020.

[357] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "DyRep: Learning representations over dynamic graphs," in *Proc. Int. Conf. Learning Representations*, New Orleans, LA, USA, May 2019.

[358] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl, and C. Leiserson, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," in *Proc. AAAI Conf. Artificial Intell.*, New York, NY, USA, Feb. 2020, pp. 5363–5370.

[359] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *ArXiv e-prints: arXiv 2006.10637*, Jun. 2020.

[360] J. Sun, W. Guo, D. Zhang, Y. Zhang, F. Regol, Y. Hu, H. Guo, R. Tang, H. Yuan, X. He, and M. Coates, "A framework for recommending accurate and diverse items using Bayesian graph convolutional neural networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. & Data Mining*, Virtual, Aug. 2020, p. 2030–2039.

[361] S. Pal, A. Valkanas, F. Regol, and M. Coates, "Bag graph: Multiple instance learning using Bayesian graph neural networks," in *Proc. AAAI Conf. Artificial Intell.*, Virtual, Feb. 2022.

[362] J. Fu, W. Zhou, and Z. Chen, "Bayesian Spatio-Temporal Graph Convolutional Network for Traffic Forecasting," *ArXiv e-prints: arXiv 2010.07498*, Oct. 2020.

[363] A. Corenflos, J. Thornton, G. Deligiannidis, and A. Doucet, "Differentiable particle filtering via entropy-regularized optimal transport," in *Proc. Int. Conf. Machine Learning*, Virtual, Jul. 2021.

[364] A. Ścibior and F. Wood, "Differentiable particle filtering without modifying the forward pass," *ArXiv e-prints: arXiv 2106.10314*, Jun. 2021.

[365] L. Tierney, "Markov chains for exploring posterior distributions," *Ann. Statist.*, vol. 22,

no. 4, pp. 1701–1728, Dec. 1994.