TraDWin: An interactive Digital Twin for City Traffic

Report submitted in fulfillment of the requirements for the Master's Thesis of

Fifth Year Integrated Dual Degree

by

Shreyansh Yadav

Under the guidance of

Dr. Vignesh Sivaraman

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI

Varanasi 221005, India

May 2024

Certificate

Copyright Transfer Certificate

Title of the Thesis: Urban Traffic Digital Twin using Physics Informed Wasserstein Generative Adversarial Nets

Name of the Student: Shreyansh Yadav

COPYRIGHT TRANSFER

The undersigned hereby assigns to the Indian Institute of Technology (Banaras Hindu University), Varanasi, all rights under copyright that may exist in and for the above thesis submitted for the award of the Integrated Dual Degree (B.Tech.+ M.Tech.) in Computer Science and Engineering.

Date: 10 May, 2024 Place: Varanasi, Uttar Pradesh Signature of the Student

Note: However, the author may reproduce or authorize others to reproduce material extracted verbatim from the thesis or derivative of the thesis for the author's personal use, provided that the source and the Institute's copyright notice are indicated.

Acknowledgments

I would like to express my sincere gratitude to Dr. Vignesh Sivaraman for his guidance and support. I sincerely appreciate his efforts in guiding me throughout this semester. I will always be thankful to him for all his effort to make this work easy and effective, which finally led to its completion.

Place: IIT (BHU) Varanasi

Date: 10 May, 2024 **Shreyansh Yadav**

Abstract

In the current era marked by rapid urbanization and population growth, cities worldwide face new challenges, chief among them being the effective management of traffic congestion and the optimization of transportation infrastructure. With reliable real-time data available through the emerging potential of smart cities powered by 5G and the Internet of Things (IoT), use of sophisticated cypherphysical systems like Digital Twins is becoming increasingly viable. Such systems, powered by ready availability of large amounts of data and machine learning methods can prove to be increasingly effective at modelling the dynamics of urban traffic flow in realtime.

In this paper, we present the implementation of such a Traffic Digital Twin (TDT), designed to seamlessly aggregate streaming traffic data from a myriad of sources including traffic sensors, cameras, GPS devices, and more. Through the integration of physics-informed deep machine learning models, our TDT serves as an alternative to traditional empirical methods of slow and resource-intensive large-scale micro-simulations. By incorporating road network information, traffic volumes at different points, and other semantic information such as road characteristics and weather, the interactive TDT is designed and tested for three crucial aspects related to urban traffic state estimation: (i) prediction of future traffic states (ii) imputation of traffic state information on missing links, and (iii) traffic re-assignment on changing map topology, which can be useful for scenarios such as road closures or newl construction planning. We test and validate our TDT using traffic volume data across Dublin city

obtained from the SCATS traffic management system and also through SUMO-based micro-simulations of relevant scenarios.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	**Description**
n_i	Number of packets in ith flow
b_i	Number of bytes transferred in the ith flow
d_i	Duration of the ith flow
μ_i	Centroid of the i-th cluster
C_i	Denotes the ith cluster
$\|\| \cdot \|\|$	Euclidean distance
F_i	Login failure of ith flow
E_i	Total login failures in ith event
Cu_i	Cumulative sum of login failures till ith event
W_i	Denotes the ith window for the LSTM model
T	Threshold to detect distributed brute force SSH attack
δ	Standard deviation of login failures in no-attack scenario
A_{IP}	anomaly score of IP address
L_F	Number of login failures for a specific IP
L_S	Number of login success for a specific IP
$A_{IP-Pref}$	anomaly score of IP prefix
L_{F-Pref}	Number of login failures for an IP prefix
L_{S-Pref}	Number of login success for an IP prefix

Abbreviations

Abbreviations	**Description**
LSTM	Long Short Term Memory
SSH	Secure Shell
RNN	Recurrent Neural Network
ML	Machine Learning
CNN	Convolutional Neural Network
PCAP	Packet Capture
RELU	Rectified Linear Unit
IP	Internet Protocol
CUSUM	Cumulative Sum

Chapter 1

Introduction

Today's rapid urbanization underscores the critical challenge of optimizing traffic and planning city infrastructure around it. Numerous studies have explored the direct and indirect impacts of traffic on health[**levy2010evaluation**], financial[**gorea2016financial**] and social domains[**anciaes2017social**]. Hence, research into intelligent systems that can assist in this direction is imperative.

The Digital Twin, as a recent development within cyber-physical systems (CPS), has garnered increased attention in recent decades[**guo2017mobile**][**singh2021digital**], particularly today with the advancements in big data, IoT connectivity, and affordable computing, which make such systems increasingly practical today. A Digital Twin is a virtual representation of a real-world object[**VANDERHORN2021113524**], system, or process, meticulously designed to replicate its physical counterpart in the digital realm. This allows it to capture and simulate the intricate details of a physical entity, enabling real-time monitoring, analysis, and prediction of its behavior. It goes beyond mere 3D modeling by incorporating live data, sensor inputs, and advanced analytics, providing a dynamic and interactive digital mirror of the physical world[**VANDERHORN2021113524**]. This capability has found applications across a multitude of fields, from manufacturing and engineering to healthcare, urban planning, and beyond.

Our traffic digital twin, TraDWin, as referred to in this paper, is an interactive simulation of city traffic that continuously updates its internal state with real-time traffic data from various sources. With the rise of smart cities equipped with IoT sensors and live camera feeds, there are numerous existing methods and theoretical approaches to enhance this capability. For instance, the Sydney Coordinated Adaptive Traffic System (SCATS)[**scats**], initially developed in Australia, utilizes inductive loop-based sensors at traffic signals to monitor traffic volume. It's already operational in over 180 cities across 28 countries, including New Zealand, Dublin, Shanghai, and Hong Kong [**wiki:sydney˙traffic˙system**]. Similarly, cities like New York, Los Angeles have devised their own systems similar to SCATS. Other approximate techniques involve traffic probes[**zhu2012probe**], mobile phone with GPS[**rose2006mobile**]—exemplified by Google's provision of congestion and travel time data—and the use of deep learning computer vision models with cameras to identify and count vehicles passing through roads. However, it's important to note that these methods primarily provide absolute volume count numbers at different nodes, as compared to collecting more granular data, such as source-destination pairs, which poses logistical and privacy challenges. In our study, we rely solely on absolute traffic volume data, without other information related to individual vehicles.

In our paper, we aim to provide an end-to-end framework for challenges of real-time data collection and aggregation. Along with a model to delve that aims to solve three key tasks related to traffic modeling which are as follows:

(i) *Traffic Prediction:* Given a traffic volume time series, our goal is to forecast future values, essentially predicting future dynamics based on historical data.

(ii) *Imputation:* Given a traffic volume time series with missing data, which could be due to sensor failure etc., fill in the missing values as accurately as possible.

(iii) *Re-assignment on edge addition or removal:* Predicting how traffic flow will be affected by changes in the road network such as adding or removing a road segment.

In contemporary literature review as we discuss later, we see mathematical as well as deep learning time series analysis methods are prevalent for tasks (i) and (ii). However, for task (iii), microscopic traffic simulation softwares like SUMO[**sumo**] and Vissim[**vissim**] are commonly employed. Such simulation engines are resource-intensive and require comprehensive information on vehicle types, sources, and destinations to yield accurate results, which may not be easily collected. Another general limitation of previous approaches which focus only on timeseries analysis is that all computation is done only taking into account traffic flow data ignoring any possible exogeneous variables like weather[**weather**], holidays[**holiday**] etc.

In our research, we assert that traffic dynamics are influenced not only by intrinsic relations within the data but also by extrinsic factors[**weather**][**holiday**] and hidden relationships may exist among these features, which are not immediately apparent. Thus the TDT model we propose is very easily extensible to include new features extending model prediction capabilities.

For the model, we use a physics informed Wasserstein based GAIN which is a modified Generative Adversarial Imputation Net (GAIN)[**gain**] (a class of conditional GANs) with modifications from Wasserstein GAN (WGAN)[**wgan**] which have been shown to be very effective in training GANs[**wgan**]. It also includes a modified loss function from physics informed deep learning techniques[**pidl**], which helps the model to conform to the physical laws of conservation.

We use Node2Vec[**node2vec**] to convert graph nodes to feature embeddings representing their structure and then append relevant traffic data and other semantic information like weather after encoding them using Word2Vec[**word2vec**] to the embeddings.

We train and test our model on real world traffic data obtained from the SCATS[**scats**] system in Dublin city and also on specific scenarios generated and simulated in SUMO[**sumo**] (Simulation of Urban Mobility) simulations.

Finally, we summarise the contributions of our study as follows:

(i) Proposing an end-to-end workflow of an interactive Traffic Digital Twin system with traffic volume data as input.

(ii) Developing and testing a new physics-informed Wasserstein distance based GAIN model to predict and model different tasks on real-world Dublin city traffic data.

(iii) Exploring physics-informed models to effectively model physical phenomena related to traffic modeling.

Section I gives the introduction of our work, Section II explores the existing research work done related to our work. Section III explains the methods used along with the implementation details to replicate the model. It also compares our approach with the existing work in this direction. Section IV presents the experiment details and the results of the proposed methods and finally Section V concludes the report.

Chapter 2

Related Work

In this section, we briefly go over the existing work done related to traffic modeling, primarily existing spatiotemporal modeling methods, both deep learning and mathematical, and compare them with our approach. We address contemporary work done for each of the different tasks that our model aims to perform, i.e., prediction, imputation, and prediction on changing topology, which is unique to our paper. This comprehensive review sets the foundation for understanding the advancements and gaps in the current methodologies, thereby highlighting the significance of our proposed approach.

2.1 Traditional Time Series Models

2.1.1 Autoregressive Integrated Moving Average (ARIMA)

Autoregressive Integrated Moving Average (ARIMA) [**arima**] and its variants, such as time-oriented ARIMA [**time˙arima**] and Seasonal ARIMA [**sarima**], are extensively employed traditional algorithms for prediction and forecasting tasks. ARIMA operates on time series data and is often combined with other algorithms to incorporate spatial features. For example, C. Xu et al. integrated ARIMA with a genetic algorithm to estimate upcoming traffic flow on roads. While ARIMA models are adept at capturing linear time series data, the addition of genetic algorithms enhances their capability to extract features from nonlinear historical data. This integration underscores the versatility and adaptability of ARIMA-based approaches in addressing complex prediction challenges. Hence, we also compare our model performance with the ARIMA model.

2.2 Matrix and Tensor Factorization Methods

2.2.1 Temporal Regularized Matrix Factorization (TRMF)

Other methods like Temporal Regularized Matrix Factorization (TRMF) [**trmf**], Bayesian Temporal Regularized Matrix Factorization (BTRMF) [**btrmf**], Bayesian Temporal Matrix Factorization (BTMF), and Bayesian Temporal Tensor Factorization (BTTF) have also been explored in the context of traffic prediction. These methods extend the principles of matrix factorization and tensor factorization to capture temporal dependencies and spatial correlations in high-dimensional traffic data. TRMF [**trmf**], introduced by Lai et al. (2017), extends TRMF to tensor data structures, allowing for the modeling of multi-dimensional traffic flow data with enhanced flexibility.

2.2.2 Bayesian Approaches

Similarly, BTRMF [**btrmf**], proposed by Liu et al. (2019), integrates Bayesian inference into tensor factorization to provide probabilistic predictions and uncertainty quantification in traffic forecasting tasks. On the other hand, BTMF and BTTF extend the Bayesian framework to matrix and tensor factorization, respectively, offering robust approaches for modeling temporal dynamics and spatial interactions in traffic networks while accounting for uncertainties in the prediction process. These methods contribute to the diverse landscape of predictive models for traffic prediction, each offering unique advantages and insights for tackling the challenges of forecasting traffic flow accurately.

2.3 Machine Learning Approaches

2.3.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks [**lstm**] are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data, making them particularly well-suited for traffic prediction tasks. LSTMs address the vanishing gradient problem prevalent in traditional RNNs by introducing a gating mechanism that regulates the flow of information. This allows LSTMs to maintain and update relevant information over long sequences, enabling them to effectively model temporal dependencies in traffic data.

2.3.2 Applications in Traffic Prediction

LSTMs have been applied successfully in various traffic prediction scenarios. For instance, Ma et al. utilized LSTMs to predict short-term traffic flow, demonstrating their ability to capture complex temporal patterns and improve prediction accuracy compared to traditional methods. Additionally, hybrid models combining LSTMs with other machine learning techniques, such as convolutional neural networks (CNNs), have been developed to leverage spatial correlations in traffic data alongside temporal dependencies. These hybrid models have shown enhanced performance in predicting traffic flow, highlighting the versatility and effectiveness of LSTM-based approaches in traffic modeling.

2.4 Graph-Based Methods

2.4.1 Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) [**gnn**] have emerged as a prominent tool for spatiotemporal modeling, demonstrating effectiveness in various applications such as traffic forecasting and imputation. For instance, studies by Li et al. (2018) and Zhang et al. (2019) have showcased their utility in accurately predicting traffic patterns and filling in missing data in traffic volume time series. GNNs operate by iteratively updating node representations based on their local neighborhood information, allowing them to capture complex spatial and temporal dependencies within graph-structured data.

2.4.2 Limitations of Traditional GNNs

However, a limitation of traditional GNNs is their reliance on the static network topology during training, which poses challenges when adapting them to dynamic graphs. As a result, they may not be suitable for tasks involving changes in the graph structure, such as predicting traffic flow upon adding or removing edges, which is a crucial aspect of our research focus. Our approach addresses this gap by incorporating mechanisms to adapt to changing topologies, thus enhancing the flexibility and applicability of GNNs in dynamic traffic networks.

2.5 Physics-Informed Deep Learning (PIDL)

2.5.1 Overview of PIDL

Physics-informed deep learning (PIDL) [**raissi2017physics**] is an emerging deep learning methodology wherein a neural network undergoes training to tackle learning tasks while adhering to the principles of physics, as given by physics-based constraint equations, general nonlinear partial differential equations, etc. This way, the fundamental laws of physics are ingrained as a bias during training, and the model does not have to figure out those dependencies on its own, which allows the resultant neural network to be data-efficient.

2.5.2 Application in Traffic Modeling

For traffic modeling, PIDL serves as an effective middle ground between purely data-driven models and purely model-driven methods. Purely model-driven approaches use a set of mathematical models and prior knowledge of traffic dynamics as input to then estimate future states. They assume the model to be an accurate representation of real-world traffic; however, such an assumption fails to capture the intricate dynamics of real-world traffic. Moreover, different equally viable models exist for the same task, and while one may be better than another for specific data, generalization across models is hard.

Some examples of model-driven traffic approaches are the Lighthill-Whitham-Richards (LWR) [**lwr**] model, Cell Transmission model (CTM) [**ctm**], etc. In contrast, pure deep learning approaches require large amounts of data to learn a good generalized relation since they learn without any pre-existing information on physical relations and constraints. Hence, physics-informed deep learning, which combines both approaches by trying to bias the deep learning model to learn the physics-based results using some parameter λ, still allows enough freedom to learn more granular relations in traffic dynamics.

2.6 Summary

In conclusion, the integration of traditional, matrix and tensor factorization, graph-based, and physics-informed deep learning methods provides a comprehensive overview of the diverse approaches employed in traffic modeling. Our proposed method aims to address the limitations identified in existing techniques, particularly in handling dynamic topologies, thereby contributing to the advancement of traffic prediction and imputation tasks.

Chapter 3

Methodology

3.1 Problem Statement

A road network can be considered an undirected graph $G = (V, E)$ where V represents nodes, which is a collection of N detector nodes, i.e., $V = \{v_1, v_2, ..., v_n\}$ where $N = |V|$ and v_i is the ith detector node. E represents the $N_E = |E|$ road links connecting the detectors, and let $A \in \mathbb{R}^{N \times N}$ denote the adjacency matrix of G, i.e., $A = \{e_{ij}\}, i, j = 1$ to N where $e_{ij} = 1$ if nodes v_i and v_j are adjacent and 0 otherwise.

Let each node of graph G at time t be represented by a D-dimensional feature vector represented as $\mathbf{x}_i^t \in \mathbb{R}^D$ that represents the node embeddings generated as explained later using node embeddings obtained from Node2Vec, other meta-information encoded, and traffic volume counts. Also, let the volume of traffic at time t at node i be c_i^t.

Define feature vectors for all nodes at a particular time t as

$$\mathbf{X}_t = (\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_N^t) \in \mathbb{R}^{N \times D} \tag{1}$$

Let the number of timesteps denoted by T be a hyperparameter denoting the number of consecutive time steps under consideration. Then denote the values of all feature vectors over all nodes over T consecutive time intervals starting at some time t_0 as

$$\boldsymbol{\chi} = (\mathbf{X}_{t_0}, \mathbf{X}_{t_0+1}, \ldots, \mathbf{X}_{t_0+T-1}) \in \mathbb{R}^{T \times N \times D} \tag{2}$$

Given the aforementioned notation, the problem that we are tackling in this paper can be formally define as proposing as Traffic Digital Twin (TDT) that can perform the following tasks:

(i) **Traffic prediction:** Given graph $G(V, E)$ and a sequence $\boldsymbol{\chi}$ of feature vectors of observed historical traffic flow over T consecutive intervals, i.e., $\boldsymbol{\chi} = (\mathbf{X}_{t_0}, \mathbf{X}_{t_0+1}, \ldots, \mathbf{X}_{t_0+T-1})$, predict $\mathbf{Y}$, the traffic volume counts c_i for $i \in N$ at the next timestep t_0+T, i.e., $\mathbf{Y} = \{c_1^{t_0+T}, c_2^{t_0+T}, \ldots, c_N^{t_0+T}\}$. Henceforth, we refer to this task as task (i).

(ii) **Imputation:** Given graph $G(V, E)$ and a sequence $\boldsymbol{\chi}$ of feature vectors of observed historical traffic flow over T consecutive intervals, i.e.,

$$\boldsymbol{\chi} = (\mathbf{X}_{t_0}, \mathbf{X}_{t_0+1}, \ldots, \mathbf{X}_{t_0+T-1})$$

, a binary mask vector $\mathbf{M} \in \mathbb{R}^{N \times T}$ such that $\mathbf{M} = \{m_{ij}\}$ where $m_{ij} = 1$ if the traffic volume count at detector i at time t i.e. c_{ij} is known and 0 otherwise to indicate its missing, impute the missing c_{ij} values. Henceforth, we refer to this task as task (ii).

(iii) **Traffic assignment on edge addition/removal:** Given original graph $G(V, E)$ and the new modified graph $G'(V', E')$ with an edge e added or removed. Let ϕ be a hyperparameter denoting the number of closest neighbor nodes to the changed edge to mask out, i.e., $\mathbf{M} = \{m_i\}$ where $m_i = 1$ if the $\text{dist}(v_i, e) > \phi$ and 0 otherwise. Predict $\mathbf{Y}' = \{c'_1, c'_2, \ldots, c'_N\}$ where c'_i is the traffic volume of the detectors with the modified topology G'. Henceforth, we refer to this task as task (iii).

3.2 Data streaming framework

Reliable real-time data collection is a crucial aspect and prerequisite of a digital twin to effectively update internal parameters and synchronize with the real world. The Traffic Digital Twin (TDT), TraDWin, presented in this paper requires real-time traffic volume data as input. We aim to use absolute traffic volume counts available at a manageable low frequency of 15 minutes to one hour. While systems with more sophisticated data at a higher frequency can be proposed, we aim to keep the real-world collection infrastructure cheap, simple, and based on already existing systems around the world. In particular, we strive to cater to the following objectives:

(i) Hardware infrastructure should be cheap and easily replaceable.

(ii) Easily extendable to use a variety of different sources.

(iii) Should work with only absolute traffic counts, no other specific information like direction and vehicle types.

(iv) Should work with a medium frequency update.

There are various possible ways to collect the type of data mentioned. The most reliable would be to install inductive loop detectors embedded in road surfaces at city intersections. Such systems are already available and in use in several major cities. For example, the Sydney Coordinated Adaptive Traffic System (SCATS)[**scats**] is available in over 180 cities across 28 countries, including New Zealand, Dublin, Shanghai, and Hong Kong [**wiki:sydney˙traffic˙system**]. New York City's Adaptive Traffic Control System (ATCS), Adaptive Signal Control Technology (ASCT), SCOOT, and ACS are other such systems which are used to control traffic signal timings and optimize congestion. Such systems can be easily extended to use with our proposed model since we only use absolute traffic volume counts which such systems are capable of collecting.

Another possible data source can be surveillance cameras assisted with computer vision models[**jain2019review**]. which is already used in practice at some intersections in Shenzhen, China. Several existing studies[**asha2018vehicle**] using state-of-the-art object detection models like YOLO[**redmon2018yolov3**] demonstrate effectiveness of this method. Such a model based approach allows for cheap and effective traffic monitoring on top of existing surveillance camera infrastructure.

Other methods like using GPS-enabled mobile phones[**rose2006mobile**] to track urban traffic flow, as Google does in its Maps product, and the use of probe vehicles[**zhu2012probe**], which are vehicles equipped with detectors that may be taxis or public transport, can also be used to approximate traffic volume based on their data.

We aim for our model to work in conjunction with multiple data streams, as that different parts of the road network may be served with different data sources. We want our TDT to seamlessly integrate with these sources, so we need an aggregator of sorts to reconcile the data from the different streams and store them in a data lake. This data can then be used by the TDT model to keep its internal state in sync with real-world data.

The deployed aggregator system will have an RPC and API interface can be interacted with by the remote sensors and other data sources. Each different service can have a separate interface build for interacting with the aggregator service. The aggregator then stores all the data with relevant meta information in a datalake which is a centralized repository that allows you to store all your structured and unstructured data. We make use of a datalake as different sources might have additional information apart from just traffic volume counts and we by allowing to store other complementary information we leave possible avenues for building up further capabilities in our TDT.

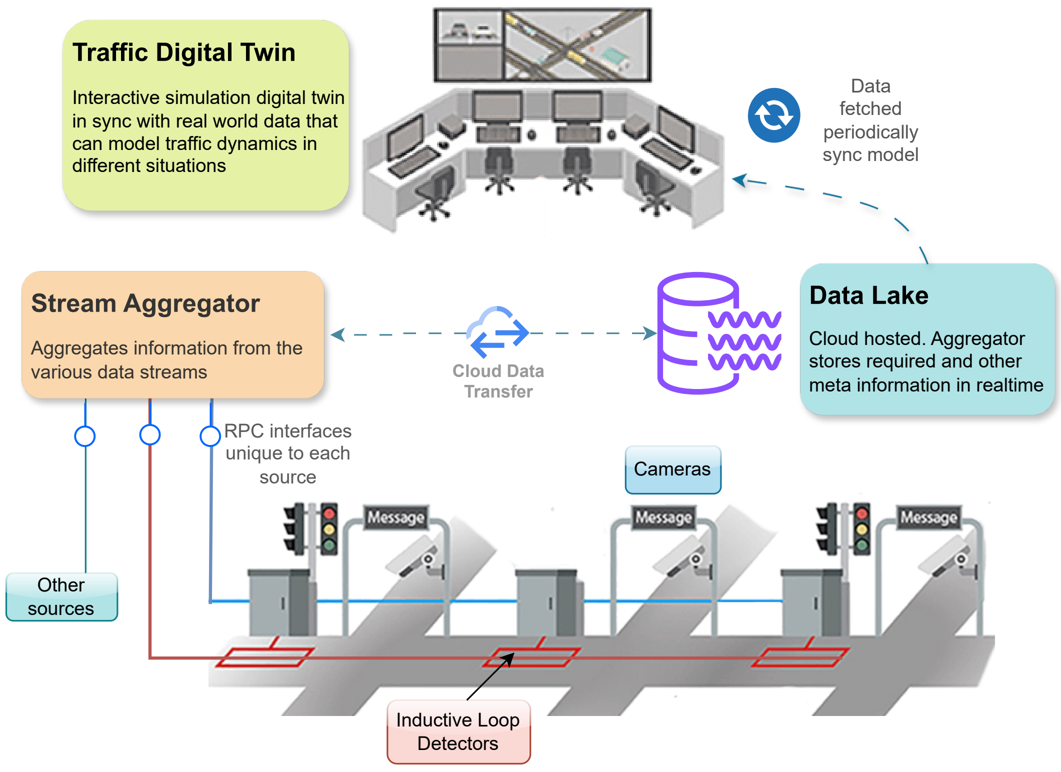

Traffic Digital Twin
Interactive simulation digital twin in sync with real world data that can model traffic dynamics in different situations
Data fetched periodically sync model
Stream Aggregator
Aggregates information from the various data streams
Cloud Data Transfer
Data Lake
Cloud hosted. Aggregator stores required and other meta information in realtime
RPC interfaces unique to each source
Cameras
Message
Message
Message
Other sources
Inductive Loop Detectors

Figure 3.1: Traffic Digital Twin (TDT) data processing

An example representing a simplied view of the aforementioned process is shown in Fig. ??.

3.3 Model input

For each node, we create a feature vector that represents all the information related to that node. The feature vector for a node is composed of three feature vectors that are as defined ahead, concatenated:

3.3.1 Graph embedding generation using Node2Vec:

In order for the model to effectively learn the spatial relationships between the detector nodes based on graph structure, we need a way to generate an appropriate representation of the graph nodes that capture the neighborhood structure effectively. $Node2Vec$[**node2vec**] is a feature learning algorithm that uses second-order biased random walks in a Skip-gram architecture to learn feature vectors for nodes in the graph. There are two key steps for feature generation:

(i) *Sampling using second order biased random walks*: Two hyper-parameters p and q are introduced which are the "return" and "in-out" parameters respectively. Suppose we simulate a random walk of fixed length l. Let c_i denote the ith node in the walk, starting with $c_0 = u$. Nodes c_i are generated by the following distribution:

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant.

For a walk that just traversed edge (t, v) and now resides at node v, we now needs to decide on the next step so it evaluates the transition probabilities π_{vx} on edges (v, x) leading from v. The transition probability $\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$, where

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

and d_{tx} denotes the shortest path distance between nodes t and x.

(ii) *Optimizing objective function using Skip-gram*: Let for graph $G(V, E)$, $f : V \to \mathbb{R}^d$ be the d-dimensional feature representation function. Thus, f is a matrix of size $|V| \times d$ parameters. Now let for every node $u \in V$, we define $N_S(u) \subset V$ as a *network neighborhood* of node u generated in step (i). Define similarity between two nodes u and v as:

$$P_f(v|u) = \frac{\exp(f(v)^T f(u))}{\sum_{w \in V} \exp(f(w)^T f(u))} \tag{1}$$

For neighborhood $N_s(v)$ of v, define the probability of the neighborhood of u as:

$$P_f(N_s(u)|u) = \prod_{v \in N_s(u)} P_f(v|u) \tag{2}$$

Further, the global neighborhood likelihood for a given f can be define as:

$$\sum_{u \in V} \log P_f(N_s(u)|u) \tag{3}$$

which is the objective function that we want to maximize through out feature representation function f. Hence,

$$\max_{u \in V} \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} \frac{\exp(f(v)^T f(u))}{\sum_{w \in V} \exp(f(w)^T f(u))} \right) \tag{4}$$

Traffic volume normalised:

The input data should be normalised to increase the training speed and effectiveness. The traffic volume counts at different detectors is normalised as:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Thus, x_{norm} is in the range $[0, 1]$ after normalization. This same approach is used for other continuous or discrete single valued features.

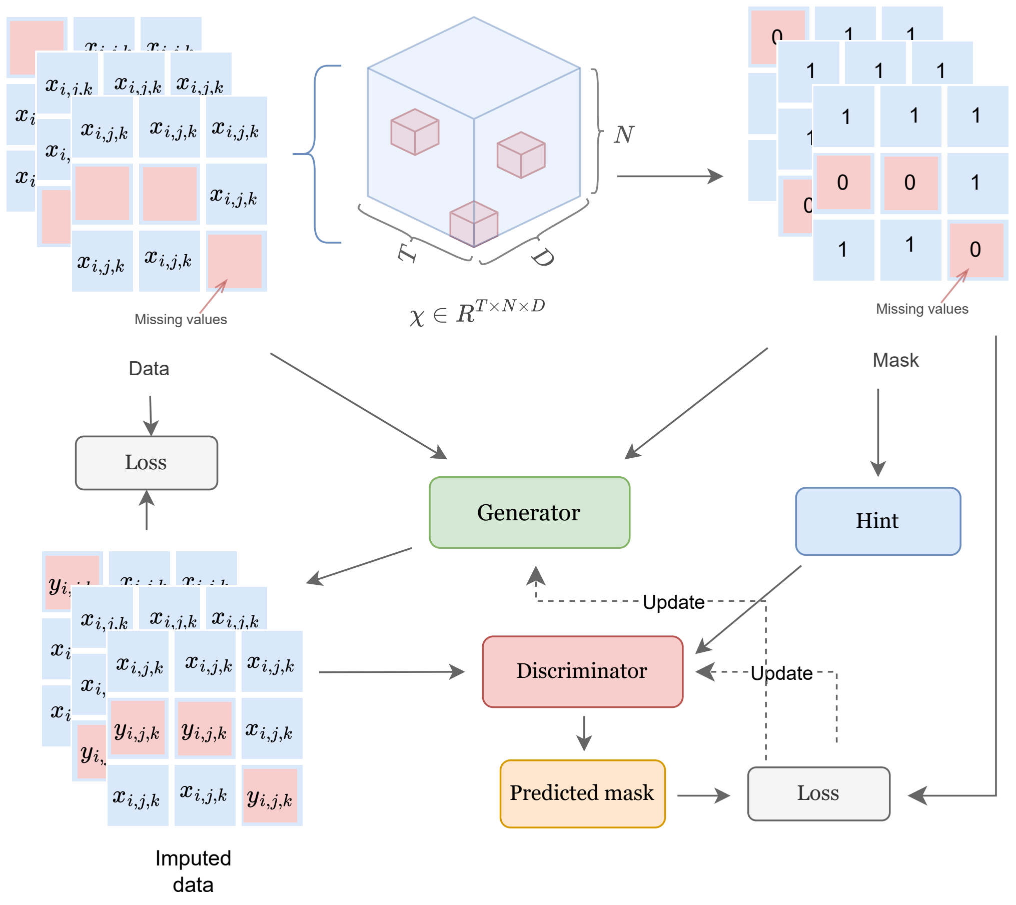


Figure 3.2: Model architecture

3.3.2 Word2Vec encoding of other exogeneous variables:

Skip-gram[**skipgram**], a popular Word2Vec[**word2vec**] architecture, aims to predict the context words given a target word. It learns to encode the semantic meaning of words by maximizing the probability of context words given the target word. Mathematically, the objective function for Skip-gram can be represented as:

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \leq j \leq n, j \neq 0} \log p \left(w_{j+1} \mid w_t \right)$$

where w_t represents the target word at position t, c is the size of the context window, and T is the total number of words in the corpus.

We use *Word2Vec* to encode the other possibly relevant exogenous variables like weather, road-type etc. as features to the model input, thus increasing its effectiveness. We would like to note that this approach is easily extensible to any number of other features that may prove to be relevant in the future, and we wish to add to the model input.

Concatenating the feature vectors from above sub-parts, we get the final input vector. Extending the notation as define in section III.A, for N nodes, we define the construction of $\mathbf{x}_i^t$, the feature vector of node i at time t. Assuming *Node2Vec* was used as defined in (i), to generate R^{D_g} dimensional graph embeddings for each node. Let g_i be the graph embedding for node i. Then, for n single-dimensional real-valued features $(r_1, r_2, \ldots, r_n)$, let them be normalized as defined in (ii) to $(r_1^{\text{norm}}, r_2^{\text{norm}}, \ldots, r_n^{\text{norm}})$ thus creating a $D_n = n$ dimensional vector. Let them be represented by h_i^t. Finally, for m "words" representing other semantic information related to the node, and using *Word2Vec* to output d_w dimensional vectors for each word, we get a R^{mxd_w} dimensioned vector that we flatten and represent as $k_i^t \in R^{D_w}$ where $D_w = m \times d_w$.

Finally, one concatenation of the above three, we get the feature vector of node i at time t

$$\mathbf{x}_i^t = (\ g_i \ || \ h_i^t \ || \ k_i^t \ ) \in R^D$$

where $D = D_g + D_n + D_w$. Thus, from (i) and (ii), over T consecutive time steps and for N nodes, we get the final feature vector as $\chi \in \mathbb{R}^{T \times N \times D}$. Depending on the task, we also add other task-specific inputs like for the case of imputation a binary matrix M representing the missing and known values. Specific implementation details are discussed further in the experiments section.

3.4 Physics Informed Deep Learning

Physics-informed deep learning (PIDL) represents a distinct approach within deep learning (DL), where a neural network is trained to tackle learning tasks while adhering to the principles of physics. By integrating fundamental physical laws as prior knowledge, the resulting neural network serves as a data-efficient approximator, adept at processing input data and delivering accurate predictions.

In situations where systems are required to adhere to the physical laws, PIDL is specially useful as it biases the model to follow the physics models, this allows the model to generate more accurate results, specially GANs as this eliminates several possible predicted distributions that are not viable do to not adhering to the physical laws of the system.

In our situation, specifically with problem 3. of Traffic assignment on edge addition/removal, where we have to predict the changed volume counts on the same timestep but with modified topology, it is imperative, according to the *conservation law* of traffic, that in such a case that the total volume of traffic in a large enough region prior to and after redistribution will be same.

Formally, for a particular fixed time, let $G(V, E)$ be the original graph, and $G'(V', E')$ be the graph obtained by adding or removing an edge e. Similarly, c_i represents the traffic volume at detector i in G for $i = 1$ to $N = |V|$, and c'_i represents the traffic volume at detector i in G' for $i = 1$ to $N' = |V'|$. Then from *conservation law*:

$$C = \sum_{i=1}^{N} c_i = C' = \sum_{i=1}^{N'} c'_i \tag{5}$$

In order to bias the model for conservation as described in Eq. (3), it needs to be integrated into the learning process of the PIDL network. The simplest way to do it is to define it as an additional term to the loss used to update the model, basically penalizing the model heavily for predictions that do not conform to the physical constraints. With traffic volumes c_i along with other graph information as training input, we establish two different measures for the discriminator to evaluate generator output:

1. $\mathcal{L}_{DL}$: Denoting the conventional discriminator loss defined as:

$$\mathcal{L}_{DL} = \mathbb{E}_{x\sim P_{\text{data}}}[D(x)] - \mathbb{E}_{z\sim P_z}[D(G(z))]$$

2. $\mathcal{L}_{PHY}$: Denoting the conservation loss. As defined in Eq. (3), for a given mask M, let S be the set of detectors such that $M(i) = 0$, specifically $S = \{i \mid M(i) = 0\}$. Then let C_0 be the sum of masked traffic volume counts, i.e., $C_0 = \sum_{i\in S} c_i$, and $Y = \{c'_i \mid i \in S\}$ be the generator output. Then define:

$$\mathcal{L}_{PHY} = (C_0 - \sum_{i\in S} c'_i)^2$$

In order to control the dominance of the different components of the loss function, we introduce two new hyperparameters, μ and λ, to adjust the weights of $\mathcal{L}_{DL}$ and $\mathcal{L}_{PH}$ respectively. Thus, the final loss function can be defined as:

$$\mathcal{L} = \mu \cdot \mathcal{L}_{DL} + \lambda \cdot \mathcal{L}_{PHY}$$

In conclusion, by leveraging hyperparameters, we can fine-tune the model's balance between different objectives. Biasing the discriminator to respect physical laws allows the generator to better capture underlying patterns in the data, a capability we found useful in our experiments as we describe ahead.

3.5 TraDWin: Traffic Model

In this section, we present the architecture of our TraDWin model, which is inspired from architectures of Generative Adversarial Imputation Nets (GAIN)[**gain**] and Wasserstein Generative Adversarial Network (WGAN)[**wgan**]. GAIN is a new and popular type of conditional GAN that has been applied to a variety of missing value imputation tasks on different datasets with favorable results. WGAN is an improvement over traditional GANs proposed by M. Arjovsky et al., that is designed to improve training stability by focusing on using Wasserstein distance metric instead of the traditional Jensen-Shannon divergence, along with some other architecture changes like gradient clipping, removal of sigmoid, etc., that have been shown to result in more reliable convergence and better quality samples.

3.5.1 *Generator:*

The generator G accepts inputs of observed data, with missing values, denoted by $\mathbf{X}$, the binary mask vector $\mathbf{M}$ which indicates what are the missing values, and a noise vector denoted by $\mathbf{Z}$. It produces imputed data $\mathbf{X}_g$, representing a vector of imputations, which is then projected to form the complete imputed output $\mathbf{Y}_0$.

Formally, the output of the generator G can be denoted as:

$$\mathbf{X}_g = G(\mathbf{X}, \mathbf{M}, (\mathbf{1} - \mathbf{M}) \odot \mathbf{Z})$$

where $\odot$ denotes the Hadamard product or the element-wise multiplication and $\mathbf{Z}$ is the d-dimensional noise vector. $\mathbf{X}_g$ represents the missing values which are then filled back in $\mathbf{X}$ as per the mask $\mathbf{M}$, and this makes the final generator output that is then passed off to the discriminator to judge.

3.5.2 *Discriminator:*

Similar to the discriminator in traditional GANs, we also use another model called the Discriminator, denoted by D, that acts as an adversary to the Generator G and trains it. The discriminator D defined in the original GAIN implementation as $D : \chi \rightarrow [0,1]^d$, outputs a binary vector instead of a single real value, which denotes what components of the generator input are real (observed) and what are fake (imputed).

So, in contrast to the traditional discriminator as used in GANs, which predicts whether the generated result is entirely fake or real, the discriminator in GAIN predicts what components are real or fake. And the loss which is then used in SGD is the mean of losses for individual components.

3.5.3 *Hint*

GAIN introduces a new concept called the hint mechanism, represented by a random variable $\mathbf{H}$ taking values in a space $\mathcal{H}$. The aim of this hint mechanism is to provide additional missing information to the discriminator about the mask. The following proposition was introduced and proved in the GAIN[**gain**] original paper:

Proposition 1 *There exist distributions of* $\mathbf{X}$*,* $\mathbf{M}$*, and* $\mathbf{H}$ *for which solutions to* $\hat{p}(\mathbf{x}|\mathbf{h}, m_i = t) = \hat{p}(\mathbf{x}|\mathbf{h})$ *for each* $i \in \{1, \ldots, d\}$*,* $\mathbf{x} \in \mathcal{X}$*, and* $\mathbf{h} \in \mathcal{H}$ *such that* $p_h(\mathbf{h}|m_i = t) > 0$ *are not unique under the optimality criterion of GAN.*

This means that there may exist several possible distributions that G may generate that may seem valid to D, so $\mathbf{H}$ must provide enough information to uniquely identify the correct representation of the underlying data, which is what the hint mechanism aims to do.

The hint mechanism depends on the binary mask vector $\mathbf{M}$, and for each imputed sample $(\hat{\mathbf{x}}, \mathbf{m})$, we draw $\mathbf{h}$ from the distribution $\mathbf{H}|\mathbf{M} = \mathbf{m}$. We then add $\mathbf{h}$ as an extra input to the discriminator, resulting in a function $D : \mathcal{X} \times \mathcal{H} \rightarrow [0, 1]^d$, where each component of $D(\hat{\mathbf{x}}, \mathbf{h})$ represents the probability that the corresponding component of $\hat{\mathbf{x}}$ was observed given $\hat{\mathbf{X}} = \hat{\mathbf{x}}$ and $\mathbf{H} = \mathbf{h}$. By varying how $\mathbf{H}$ is defined, we control the level of information it provides about $\mathbf{M}$.

Finally, we take inspiration from the architecture of WGAN and incorporate some of those changes in our model. Traditional GANs suffer from training instability and model collapse, i.e., inability to learn the diverse representations and only generating repetitive samples. Wasserstein GAN (WGAN) proposed by M Arjovsky et. al, solves these problems and we highlight some of the architectural changes we made to GAIN based on WGAN in the following paragraphs.

In traditional GAN the generator and discriminator are trained using the following loss functions:

$$\mathcal{L}^{G}_{\text{GAN}} = \log(1 - D(G(z)))$$

$$\mathcal{L}^{D}_{\text{GAN}} = -\log(D(x)) - \log(1 - D(G(z)))$$

whereas in WGANs, we remove the logarithms and use Wasserstein distance for loss instead.

$$\mathcal{L}^{G}_{\text{WGAN}} = -D(G(z))$$

$$\mathcal{L}^{D}_{\text{WGAN}} = D(G(z)) - D(x)$$

Another change in WGANs, that we incorporate in GAIN is the absence of a sigmoid activation function in the discriminator's output layer, unlike in traditional GANs. This alteration enables the discriminator to output unbounded real-valued scores instead of probabilities. Additionally, WGANs incorporate gradient clipping as a regularization technique to enhance training stability. Gradient clipping involves setting a threshold value for the gradients, and if they exceed this threshold, they are scaled down to ensure that their magnitude does not surpass it. This approach helps stabilize the training process and mitigates issues such as exploding gradients.

Chapter 4

Experiments and Results

In this section we describe the experimental results of our model along with the necessary parameters required to reproduce those results. We also describe baselines that we compare our model with, along with their parameters.

4.1 Dataset Description

We use the real world traffic volumes of Dublin city streets collected using the SCATS (Sydney Coordinated Adaptive Traffic System)[**scats**] system over a three month period. The SCATS system is used in Dublin as an adaptive urban traffic management system that synchronises traffic signals and optimises traffic flow across the entire city.

The dataset consists of traffic volumes from the 825 sensors at all key intersections in Dublin city, sampled at a frequency of 1 hour, during a time period of 3 months, from July 1 2023 to September 1, 2023. The dataset contains the latitudinal and longitudinal geographical coordinates along with timestamped total volume of traffic detected by each detector in the one hour period.

We also verify the model for task (iii) on SUMO[**sumo**] TAPASCologne scenario[**tapas**], which is a simulation configuration that describes the traffic dynamics of Cologne city in Germany for a day. The scenario is publicly available on the official SUMO website and was created using mobility demand data based on information about travelling habits of citizens and on information about the infrastructure of the area they live in. Since this testing is manual, we run a limited number of tests, where we change an edge and process the simulation, based on the default minimal travel time routing before and after the change, and observe the deviation between simulation and our results.

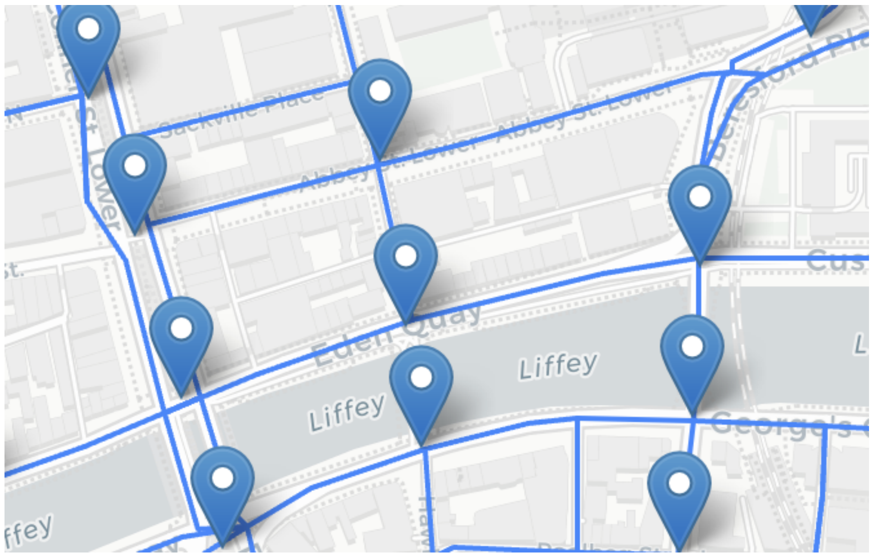


Figure 4.1: Snapshot of sensor locations plotted along with road network from Overpass API

4.2 Model Parameters

For all experiments, we set hyperparameters $T = 3$, which is the number of consecutive timesteps to consider for prediction and imputation tasks. For all tasks, we optimize the model input by assuming the locality of traffic within such a timespan and that any changes to a particular node are only dependent on nodes "close" to it. Formally, during training, we train on sub-graphs consisting of the node clusters instead of the entire graph all at once. Specifically, a subgraph is generated by choosing a node u at random and then selecting all nodes that are at a distance less than σ to it, specifically for $G(V, E)$, $G_{\text{sub}} = G\{v \in V \mid \text{dist}(u, v) < \sigma\}$. For our experiments, we set σ as 2.5 km, which is a reasonable expectation as clusters usually contain 20 to 30 nodes within this range.

After sampling as stated before, we use a 80-20 test-train split, i.e., the model is trained on 80% of the samples and then performance is verified on new unseen data that comprises of 20% of the samples. The data is then modified based on the task at hand. For imputation, we followed MCAR (Missing Completely at Random) distribution and randomly masked values both spatially and temporally, following a parameter defined as the miss rate, we test our model on miss rates of 10%, 20%, 30%, and 40%. For prediction we, use the full data of three timesteps to predict the fourth step, by basically masking the values along the last timestep dimension. For re-assignment task, i.e., task (iii), we only consider one timestep and re-assign values for that same timestep but with modified spatial geometry.

4.3 Baselines

We compare our TraDWin model with a number of different existing approaches across different domains including different mathematical analysis and deep learning techniques used popularly for these tasks.

One of the simplest approaches for imputation is *KNN*. Similarly, for prediction, *ARIMA*[**arima**] stands for AutoRegressive Integrated Moving Average is used. Matrix factorization methods like *TRMF*[**trmf**] (Temporal Regularized Matrix Factorization) use latent factors for predictions, and its modification *BTRMF*[**btrmf**] (Bayesian Temporal Regularized Matrix Factorization) extends TRMF with a Bayesian framework. For both these methods, we use rank = 10, 1000 burn-in iterations, and 200 Gibbs iterations.

LRTC-TNN[**lrtc**] (Low-Rank Tensor Completion with Truncated Nuclear Norm) is a method for tensor completion. It uses parameters $\rho = 1e-5$, $\theta = 0.25$, and $\epsilon = 1e-4$. *BGCP* (Bayesian Gaussian Process Factorization) is another method that utilizes Bayesian inference. For BGCP, we use similar burn-in and Gibbs iterations, and set rank = 30. It is worth noting that these model baselines are only applicable to task (i) and task (ii) and not for task (iii) since they were not built with that task as a consideration.

Finally, there are deep learning methods that we used. For imputation, we employed the *Denoising AutoEncoder* (DAE)[**dae**] model, which is effective in learning meaningful representations of the data while handling missing values. For prediction tasks, we utilized an *LSTM*[**lstm**] (Long Short-Term Memory) model as LSTM networks are well-suited for capturing temporal dependencies in sequential data.

4.4 Evaluation metrics

In order to evaluate the performance of the different methods and compare them, we use RMSE (Root Mean Squared Error) and MAPE (Mean Absolute Percentage Error). These metrics are defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%$$

where y_i represents the true value, $\hat{y}_i$ represents the predicted value, and n is the number of samples.

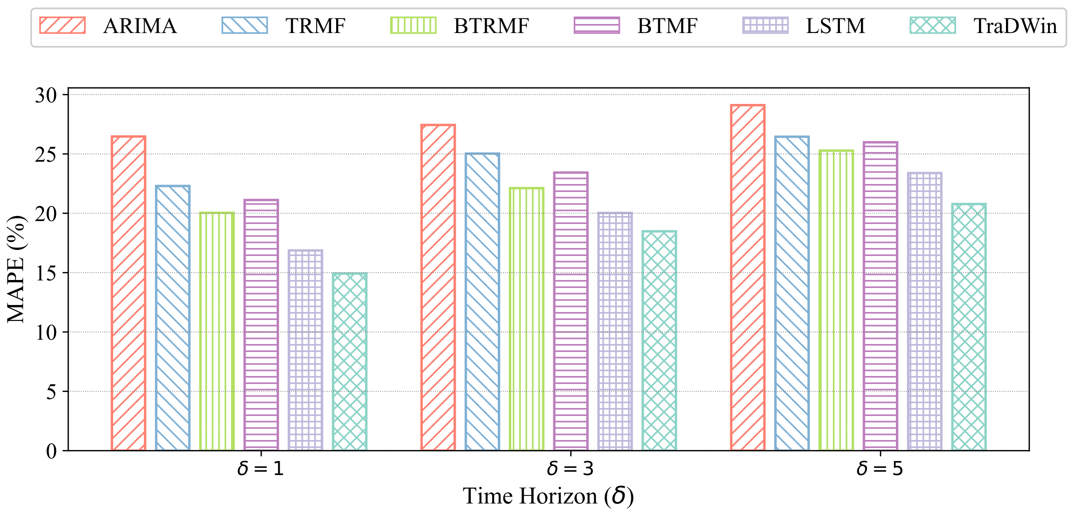
4.5 Results

We evaluated the performance of our model on different tasks and compared them with other popular models used in the domain.

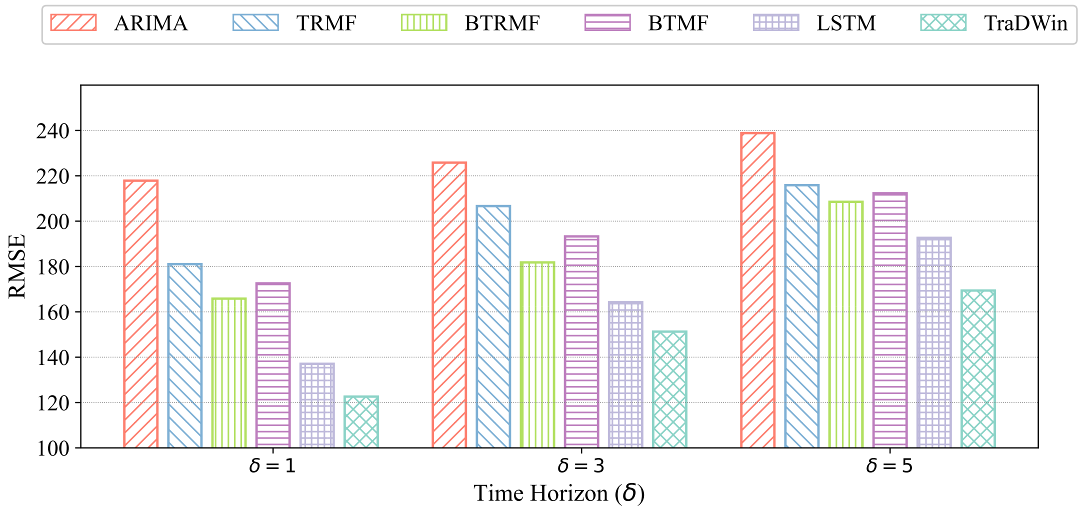
4.5.1 Prediction

For the prediction task, we tested the models for predicting traffic volumes at the next timestep, i.e., the forecasting $\delta = 1$, at the third timestep, i.e., $\delta = 3$ and the fifth timestep in future, i.e., $\delta = 5$. This scenario of predicting the next time step requires a mask consisting of all the T_{n+1} values missing, i.e., 0. We prepare a mask similarly for other time horizons. Thus, prediction here is treated as a strict MNAR (Missing Not At Random) subcategory of imputation.

The MAPE and RMSE plots of prediction task on different horizons is shown in Fig. **??** and Fig. **??** respectively. We observer that in general, as the δ i.e. the time horizon to predict in future, increases, both metrics worsen with MAPE falling 5-10% from $\delta = 1$ to $\delta = 3$, i.e., we see a stronger short term accuracy but face some challenges with long-term prediction, across all models. This trend is common in time-series forecasting, where predictive accuracy decreases as the prediction horizon extends. The primary reasons for this decline include the increasing uncertainty and the influence of unpredictable external factors, such as accidents or weather changes. We see that ARIMA performs best, which is to be expected since its the simplest of mathematical models among the baselines we consider. The other matrix factorization and Bayesian models like TRMF, BTRMF and BTMF perform better but given that they are stricly mathematical models, they (i) fail to capture the intricate traffic dynamics that a deep learning model can, and (ii) use only the timeseries traffic data and not the graph topology and external factors that our model considers like weather etc. Finally, we see that LSTMs are close with slightly (2-3%) worse performance which could be attributed to lack of knowledge of graph topology that we incorporate in our model through $Node2Vec$.



(a) MAPE on prediction task for different deltas (timesteps to predict ahead)



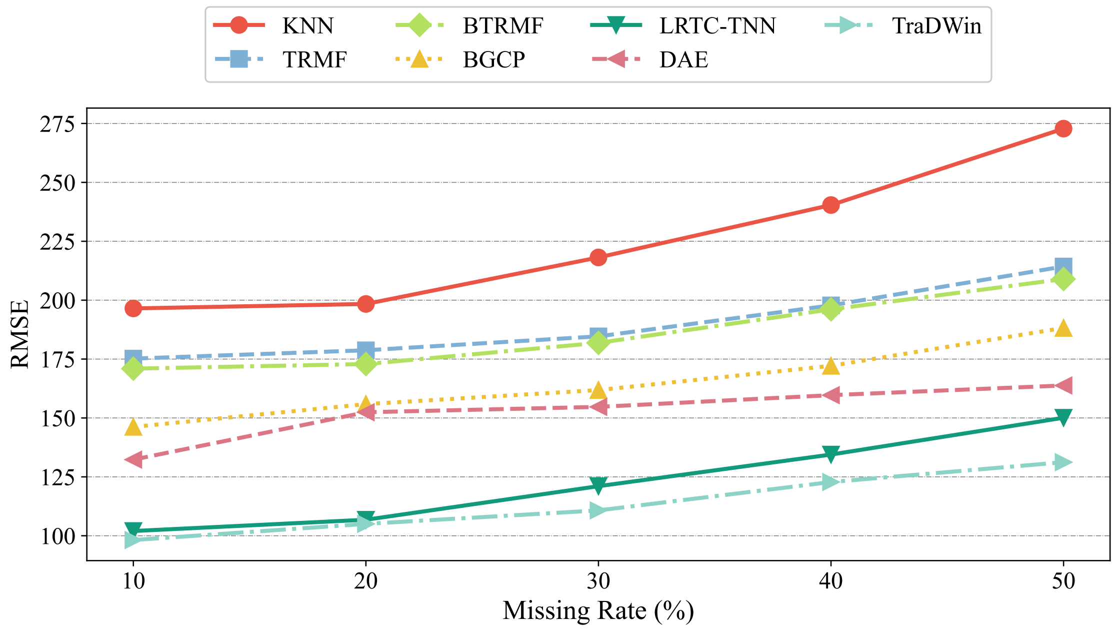
(b) RMSE on prediction task for different deltas (timesteps to predict ahead)

Figure 4.2: MAPE and RMSE on prediction task for different deltas

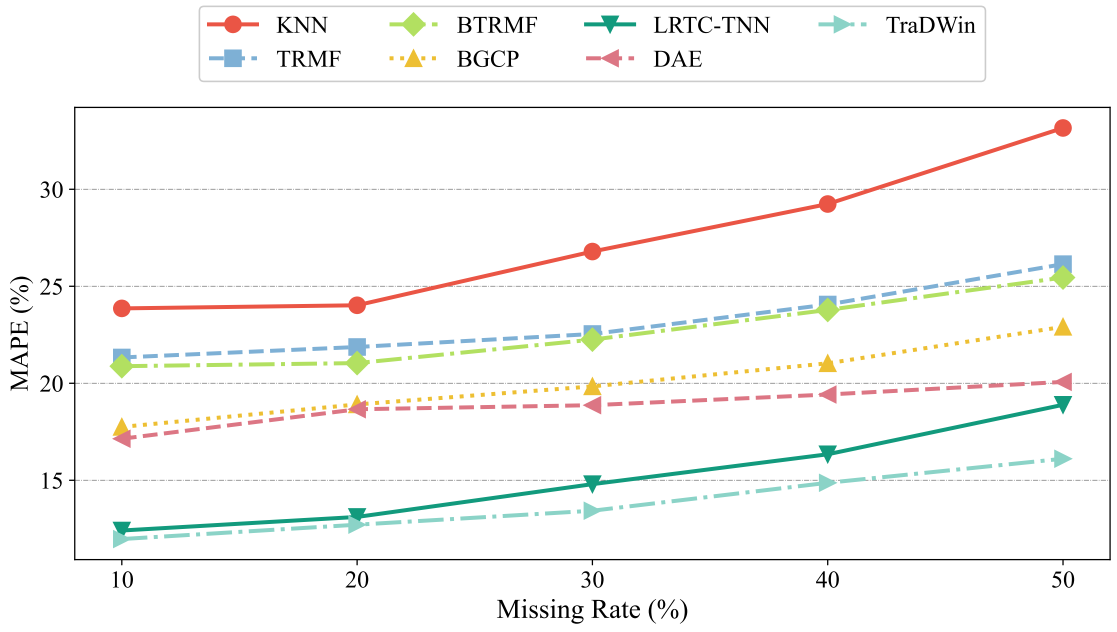
4.5.2 Imputation

Next, we evaluate performance on the imptation task. For this task, we have considered the MCAR (Missing Completely At Random) distribution and compare the models at five missing rates of 10%, 20%, 30%, 40% and 50%. The perfomance of different models measured using MAPE and RMSE is shown in Fig. **??**. We see that the conventional methods like KNN, perform poorly to more specialised approaches, which is to be expected since KNN is not enough to capture the intricate traffic dynamics. The matrix factorisation and Bayesian methods which are TRMF, BTRMF and BGCP work slightly better, with deep learning methods like DAE following ahead, though since they deal with only traffic data timeseries with no information on topology they have a 8-13% worse off performane than out model. Tensor completion based model like LRTC-TNN is close to our TraDWin in for low missing rates, but we see that it does not scale as well with increasing missing rate, possibly because too much data is lost for reliable tensor completion which a deep learning model can still handle, due to being able to learn more complex relationships in data and having knowledge of graph topology and other external factors. Moreover, it is worth nothing that LRTC-TNN is a MCAR imputer and not applicable to task (iii) so not generalised enough for our usecase. For all models, we see that the accuracy of imputations decreases with higher missing rates due to the reduced amount of contextual information available. This is a common challenge in data imputation, where fewer data points make it harder to discern underlying patterns accurately.

We also observe that TraDWin performs worse on task (i) than task (ii) by around 4%. One primary reason for this, as we suspect, could be that GAIN that was originally designed for MCAR (Missing Completely At Random) imputation, does not generalise well to MNAR imputation scenarios. Another possible reason could be the subgraph-based computation method that we use, where nodes at the corners of the subgraph do not have adequate spatial and temporal information of their neighbors. Still, we observe that our model performs reasonably and delivers comparable results for being a generalized model.



(a) RMSE of different models on imputation task



(b) MAPE of different models on imputation task

Figure 4.3: Imputation comparison for different missing rates

4.5.3 Re-assignment

Finally, on the re-assignment task, which is a new task that we address in our paper and is not conventionally seen in contemporary literature, we essentially train the model for MNAR (Missing Not At Random) imputation scenarios with the central node cluster missing. This effectively makes the model learn how to assign traffic to node clusters given neighbor node information. Based on this training task, we achieve a MAPE of 13.47% and RMSE of 109.90. Note that the other baselines considered for prediction and imputation are not applicable to this without changes to their model architecture so there is not comparison with contemporary models. Moreover, the de-facto way to solve this problem, as we have seen in our literature review, has been simulations like SUMO[**sumo**] and Vissim[**vissim**]. We too use those to compare and test our model on the TAPASCologne scenario using SUMO, which is a traffic simulation engine, with the configuration as described previously. The results of our model on this task are shown in Table. **??**. We observe that the model achieves a MAPE of 13.47% on Dublin SCATS dataset and 15.06% on the TAPASCologne scenario, which demonstrates that the model can, with reasonable accuracy, learn to solve the traffic assignment problem on node clusters. Further, we tried to evaluate how our model's performance scales with number of modifications, i.e., how much can we alter the original graph such that the model is still viable to use for re-assignment. In Fig. 7, we show a comparison of MAPE and RMSE with number of edges modified. We see that performance declines steadily from 13.47% at 1 change to 31.91% at 5 changes. So, altering more of graph leads to decline in performance, with drop being significant and sharp after 3 modifications. A possible reason for this could be the mass masking strategy we use to train the model for task (iii). For 5 modifications, too many of the neighbouring nodes are masked leading to large loss of contextual information that knowledge of graph representation does not compensate enough.

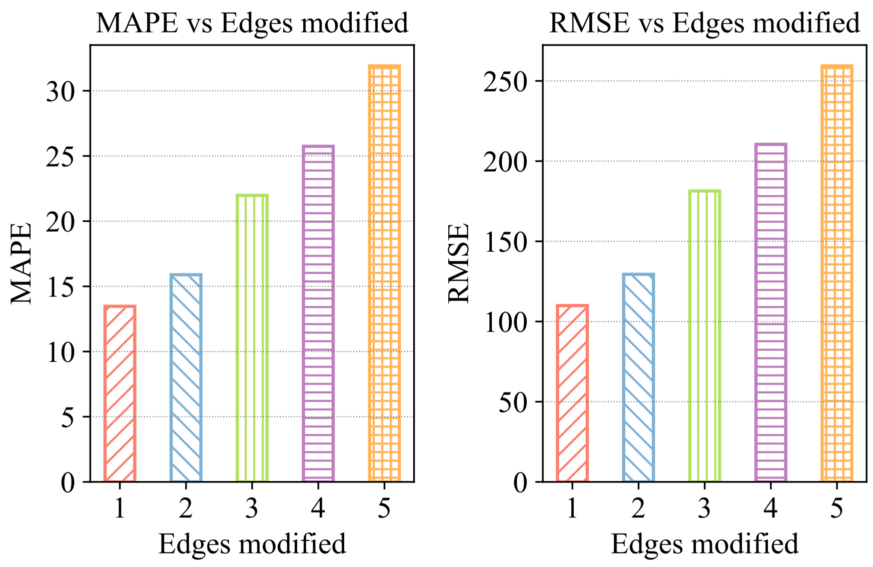


Figure 4.4: MAPE and RMSE on re-assignment task vs number of edges modified

Table 4.1: Performance on re-assignment task (1 edge change) for different datasets

Dataset	MAPE (%)	RMSE
Dublin SCATS[**dublin˙scats**]	13.47	109.90
TAPASCologne[**tapas**]	15.06	23.34

Chapter 5

Conclusion and Future Work

In conclusion, we proposed an end-to-end framework, including a model, for an interactive traffic digital twin. We also evaluated and compared our model's performance on the real-world Dublin SCATS dataset, and our experiments showed that the model produced reasonable results compared to other contemporary approaches and techniques in different scenarios. We believe that such a system would be increasingly useful for managing and guiding the decision-making processes related to traffic infrastructure and planning, both by governments and private entities.

Future Work

It is worth noting that the core architecture of our TDT model is the same for all three underlying tasks. In our approach, we deal with each task as a separate problem for the model to be trained upon. Given that multi-task learning for models, where the model is able to better generalize by learning several different tasks, with output controlled by the parameters of the input itself, has shown promising results, we believe it will be worthwhile to investigate the application of that approach to our model.