LLM Based Approaches For Traffic Prediction In Networks Traffic

M.Tech Thesis

by

Rahul Kushwah

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2025

LLM Based Approaches For Traffic Prediction In Networks Traffic

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Technology

by

Rahul Kushwah
2302102015

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2025

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **LLMs Based Approaches For Traffic Prediction In Networks Traffic.** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology** and submitted in the **Department of Electrical Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the period from July 2024 to May 2025 under the supervision of Dr. Dibbendu Roy, Indian Institute of Technology Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Rahul 26/05/25

Signature of the Student with Date

(Rahul Kushwah)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dibbendu Roy

27/5/25

Signature of Thesis Supervisor with Date

(Dr.Dibbendu Roy)

Rahul Kushwah has successfully given his/her M.Tech. Oral Examination held on 07 May 2025.

Dibbendu Roy	Saptarshi Ghosh
Signature(s) of Supervisor(s) of M.Tech. thesis	Convener, DPGC
Date: 27/5/25	Date: 30-05-2025

ACKNOWLEDGEMENTS

I would like to thank everyone who helped and supported me during my M.Tech journey and the completion of this thesis.

First of all, I am very thankful to my respected guide, **Dr. Dibbendu Roy** for his continuous guidance, support, and motivation throughout this research work. His expert advice, valuable suggestions, and encouragement helped me at every stage of the thesis. Whenever I had doubts or faced challenges, he was always there to help and guide me in the right direction.

I also want to thank all the professors and staff members of the **Electrical Engineering Department, IIT Indore** for their support and for providing me with quality education during my M.Tech. Their teachings helped me gain knowledge and confidence in my subject.

I am also very grateful to my classmates and friends who supported me during this journey. They were always there for discussions, solving problems together, and sharing experiences, which made this work more enjoyable and easier.

I would like to thank **IIT Indore** for providing excellent lab facilities, internet access, software tools, and a peaceful research environment. These resources played an important role in completing my experiments and simulations successfully.

Last but not least, I am deeply thankful to my **family** especially my parents for their endless love, support, and belief in me. Their emotional strength and encouragement helped me stay focused and motivated during tough times.

This thesis is the result of not just my efforts, but also the support and guidance of all these wonderful people. I sincerely thank each one of them from the bottom of my heart.

Rahul Kushwah
Masters of Technology
(Communication and Signal Processing)
Roll Number: 2302102015
IIT Indore

ABSTRACT

In modern communication networks, particularly within the context of 5G and beyond, network slicing has emerged as a key technique to support diverse services with varying Quality of Service (QoS) requirements. Each slice is designed to meet the specific needs of applications such as video streaming, IoT, and ultra-reliable low-latency communications, and must be provisioned with appropriate resources.

A major challenge in network slicing is the dynamic and unpredictable nature of network traffic. As traffic is user-generated and varies over time, it cannot be directly controlled by the network operator. This time-varying behavior makes static resource allocation strategies inefficient, potentially leading to congestion, increased delay, or poor resource utilization. Therefore, accurate traffic prediction is essential to enable proactive and adaptive resource management.

This thesis investigates the application of deep learning techniques for traffic prediction in network slicing scenarios. Specifically, Long Short-Term Memory (LSTM) and Transformer-based models are explored due to their ability to capture long-term temporal dependencies in time-series data. The study is carried out using classical single-server queuing models such as M/M/1, D/D/1, M/G/1, G/G/1, and G/M/1, which help in understanding basic traffic behaviors in a controlled setting.

However, it is important to note that real-world traffic conditions are more complex. At router and base station sites, traffic from multiple services is often multiplexed, resulting in aggregated arrival patterns that do not follow regular or idealized distributions. These mixtures of heterogeneous traffic types make the arrival statistics highly irregular and challenging to model using traditional analytical methods. In such scenarios, data-driven approaches like deep learning provide a promising solution by learning complex patterns directly from empirical traffic data.

The predicted traffic is then utilized to guide dynamic and intelligent resource allocation, enhancing the adaptability and efficiency of the network slicing framework.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

With the exponential growth of mobile devices, IoT systems, and internet connected applications, modern communication networks face increasing pressure to deliver reliable, highspeed data services [25] . The proliferation of multimedia content, cloud-based applications, and real-time services such as video conferencing and gaming necessitates precise and efficient traffic management strategies. Technologies such as 5G and the forthcoming 6G networks introduce stringent requirements for low latency, high throughput, and ultra-reliable communication. These include key use cases such as ultra-reliable low-latency communication, massive machine type communication, and enhanced mobile broadband. One of the most promising techniques to address these complex requirements is network traffic prediction. It enables operators to forecast future data load using historical traffic patterns, allowing them to dynamically allocate network resources, avoid congestion, and optimize quality of service (QoS) [27] . Effective traffic prediction reduces packet loss, improves bandwidth utilization, and supports proactive rather than reactive network management.

1.2 Motivation

Predicting network traffic is essential for keeping communication networks running smoothly. When we can accurately forecast traffic patterns, it becomes easier to manage bandwidth, reduce delays, and provide users with a better overall experience. Traditional methods like ARIMA, moving averages, and exponential smoothing have been used for prediction, but they often fall short. These models struggle with the complex, non-linear patterns and long-term trends that are common in real-world network traffic. In recent years, machine learning especially deep learning—has opened new doors. Models like Long Short-Term Memory (LSTM) networks and Transformers have shown great success in predicting time-series data [22, 13]. LSTM networks are built to understand sequences and long term relationships, while Transformers use self attention to identify important patterns over time [29]. By using these advanced models in network traffic prediction, we can make forecasts that are more accurate. This helps network providers act before problems occur—like traffic spikes—so they can better allocate resources. As a result, the network becomes more efficient, scalable, and reliable for everyone who uses it.

1.3 Problem Statement

Modern networks face increasing traffic complexity due to varying data loads, heterogeneous devices, and dynamic user behavior [25]. Traditional forecasting methods often fall short in capturing the stochastic and temporal characteristics of such data. Consequently, there's a need for sophisticated deep learning-based approaches that can learn intricate patterns from time-series data and provide accurate forecasts.

This research aims to explore and compare two deep learning models—LSTM and Transformer for network traffic prediction using Python-based simulation. The study focuses on evaluating model performance on aggregator of different types of queueing models (such as M/M/1, D/D/1, etc.) using synthetic and real traffic datasets. It seeks to identify the most effective approach for predicting key performance indicators such as arrival time, service time, latency, and jitter.

1.4 Objectives

The primary objectives of this research are:

- To collect and preprocess network traffic data representing aggregate various queueing scenarios.
- To implement deep learning models, specifically LSTM and Transformer, for forecasting network metrics.
- To evaluate the models based on performance metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R Squared (R^2) score [36].
- To analyzed and compared the performance of LSTM and Transformer architectures.
- To visualize and interpret the predicted trends for network traffic parameters.
- To derive insights that could support more efficient resource allocation strategies in future communication systems.

1.5 Scope of the Work

This study is focused on software-based modeling and analysis of network traffic prediction. The scope is restricted to using Python for the development, training, and evaluation of LSTM and Transformer models. Experiments are conducted using both synthetic datasets based on queueing models and real-world datasets where applicable.

Key libraries are used include NumPy for numerical operation, Pandas for data manipulation, TensorFlow and Keras for deep learning, and Matplotlib/Seaborn for data visualization. The research does not cover hardware implementation, integration with live systems, or physical layer modeling. However, the outcomes provide a theoretical and practical foundation for such future work.

1.6 Significance of the Study

This research contributes to the growing field of intelligent network management by applying state of art deep learning techniques to the problem of traffic prediction. Accurate forecasting supports improved decision-making in network planning, including dynamic resource provisioning, congestion avoidance, and service quality improvement.

Furthermore, the comparative analysis of LSTM and Transformer models offers insights into their strengths and limitations in handling different traffic patterns. The findings may inform the development of hybrid or ensemble models, adaptive systems, and real-time predictive engines. As networks evolve toward 6G and beyond, such predictive capabilities will become increasingly vital.

Chapter 2

Literature Review

2.1 Advancements in Communication Technologies

The rapid evolution of communication technologies, especially the transition from 5G to 6G, has intensified research efforts focused on enhancing the efficiency of network traffic prediction, resource allocation, and network slicing orchestration. As the demands of modern applications grow more complex, the networking community has turned to artificial intelligence (AI) for innovative solutions. Among the most promising approaches are those based on Large Language Models (LLM) and Transformer-based architectures [13, 14], which bring powerful capabilities in natural language understanding and deep representation learning to the real of networking.

2.2 Large Language Models in Network Management

ChatNet: LLM for Automated Network Intelligence (Huang et al., 2023) Huang et al. presented ChatNet, a pioneering framework designed to bridge the gap between human intent and network operations through the use of LLM [11]. This model leverages techniques such as prompt engineering and parameter-efficient finetuning to adapt general-purpose LLMs to the specific linguistic and operational demands of networking environments. ChatNet is built on the concept of embodied intelligence, where LLMs are not only capable of understanding language but also of interfacing with network tools such as solvers, analyzers, and visualization platforms [16]. This framework allows for the execution of complex tasks like capacity planning, fault diagnosis, and security enforcement by interpreting natural language queries and mapping them to network actions. Despite its strengths, ChatNet remains predominantly focused on semantic transformation and tool orchestration, lacking the infrastructure for predictive modeling based on time-series traffic data, which is vital for proactive resource allocation in dynamic network environments.

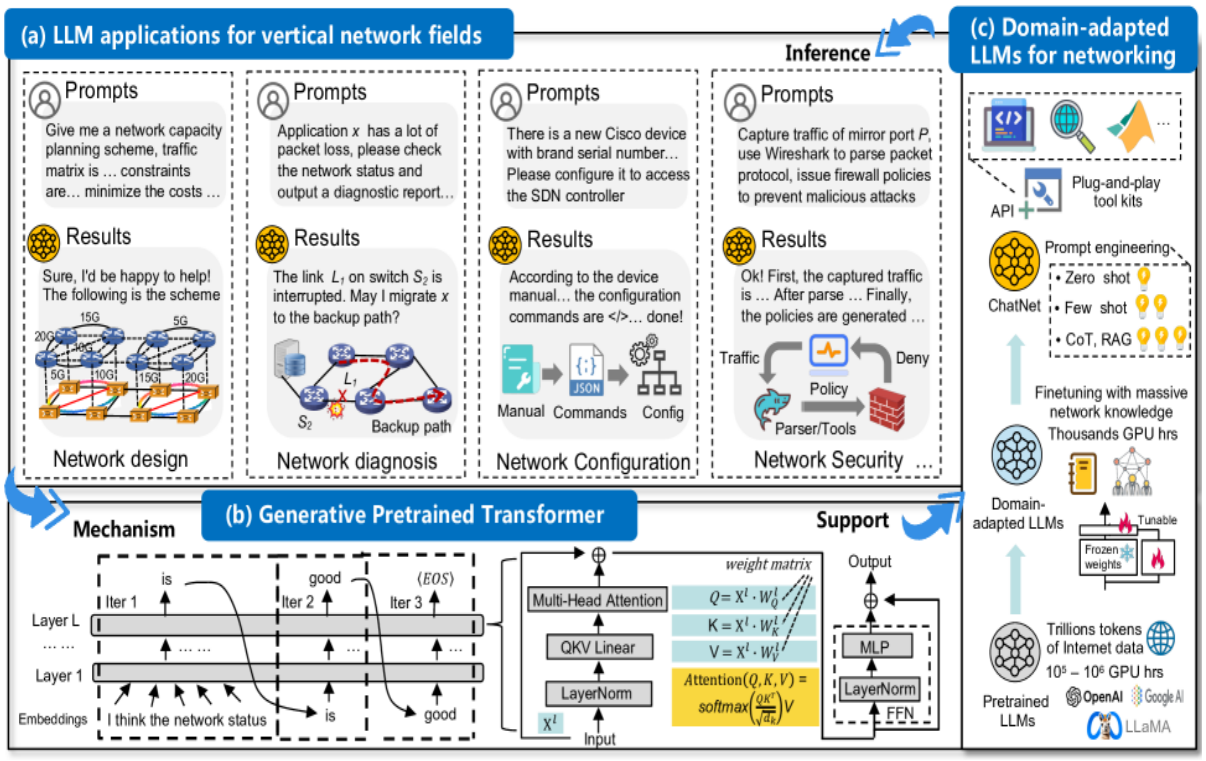

(a) LLM applications for vertical network fields
Inference
(c) Domain-adapted LLMs for networking
Prompts
Give me a network capacity planning scheme, traffic matrix is … constraints are… minimize the costs …
Results
Sure, I'd be happy to help! The following is the scheme
Network design
Prompts
Application x has a lot of packet loss, please check the network status and output a diagnostic report…
Results
The link L_1 on switch S_2 is interrupted. May I migrate x to the backup path?
Backup path
Network diagnosis
Prompts
There is a new Cisco device with brand serial number… Please configure it to access the SDN controller
Results
According to the device manual… the configuration commands are &lt;/&gt;… done!
Manual Commands Config
Network Configuration
Prompts
Capture traffic of mirror port P, use Wireshark to parse packet protocol, issue firewall policies to prevent malicious attacks
Results
Ok! First, the captured traffic is … After parse … Finally, the policies are generated …
Traffic Policy Deny
Parser/Tools
Network Security …
API
Plug-and-play tool kits
ChatNet
Prompt engineering
• Zero shot
• Few shot
• CoT, RAG
Finetuning with massive network knowledge
Thousands GPU hrs
Domain-adapted LLMs
Tunable
Frozen weights
Trillions tokens of Internet data
$10^5 - 10^6$ GPU hrs
Pretrained LLMs
OpenAI
Google AI
LLaMA
Mechanism
(b) Generative Pretrained Transformer
Support
Iter 1 is
Iter 2 good
Iter 3 ⟨EOS⟩
Layer L
Layer 1
Embeddings
I think the network status is good
Multi-Head Attention
QKV Linear
LayerNorm
Input
weight matrix
$Q = X^l \cdot W_Q^l$
$K = X^l \cdot W_K^l$
$V = X^l \cdot W_V^l$
$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
Output
MLP
LayerNorm
FFN

Figure 2.1: Applications, mechanisms, and enabling techniques for domain-adapted network LLM.

[11]

Multi-Agent LLM Framework for Network Slicing Orchestration (Dandoush et al., 2024)

In their work, Dandoush et al. advanced the conversation by integrating LLMs into a multi-agent system for the orchestration and management of network slicing [12]. This framework envisions a decentralized model where multiple LLM agents operate within different network domains (e.g., access, core, cloud) and collaborate to interpret user intents, allocate resources, and monitor slice performance. Each agent is capable of translating high-level user goals into detailed slice configurations using standard descriptors and orchestrating actions based on abstracted views of the infrastructure [15]. One of the significant contributions of this work is its approach to translating qualitative user inputs into quantifiable service parameters and using these to automate end-to-end slice lifecycle management. However, while the system excels in adaptability and intent interpretation, it does not explicitly incorporate predictive analytics or numerical forecasting mechanisms needed to anticipate traffic loads or optimize resource provisioning dynamically [17].

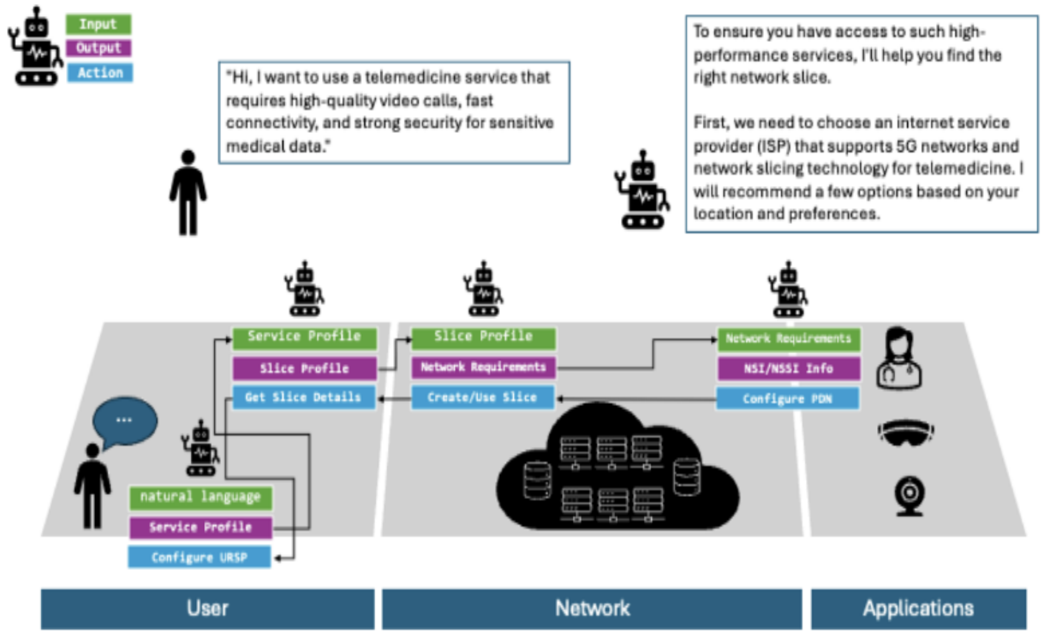
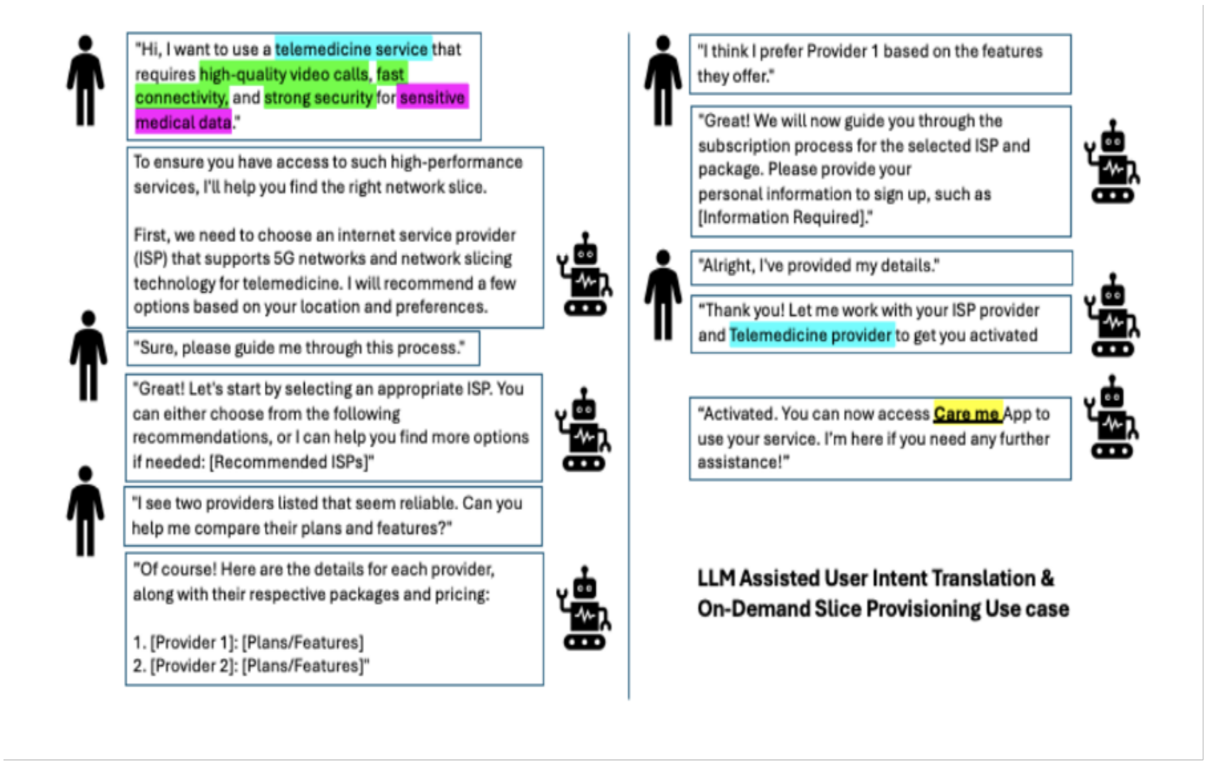

Input
Output
Action
"Hi, I want to use a telemedicine service that requires high-quality video calls, fast connectivity, and strong security for sensitive medical data."
To ensure you have access to such high-performance services, I'll help you find the right network slice.
First, we need to choose an internet service provider (ISP) that supports 5G networks and network slicing technology for telemedicine. I will recommend a few options based on your location and preferences.
Service Profile
Slice Profile
Get Slice Details
Slice Profile
Network Requirements
Create/Use Slice
Network Requirements
NSI/NSSI Info
Configure PDN
...
natural language
Service Profile
Configure URSP
User
Network
Applications

Figure 2.2: LLM Assisted End User Network Slice Intent Translation. [12]



"Hi, I want to use a telemedicine service that requires high-quality video calls, fast connectivity, and strong security for sensitive medical data."

To ensure you have access to such high-performance services, I'll help you find the right network slice.

First, we need to choose an internet service provider (ISP) that supports 5G networks and network slicing technology for telemedicine. I will recommend a few options based on your location and preferences.

"Sure, please guide me through this process."

"Great! Let's start by selecting an appropriate ISP. You can either choose from the following recommendations, or I can help you find more options if needed: [Recommended ISPs]"

"I see two providers listed that seem reliable. Can you help me compare their plans and features?"

"Of course! Here are the details for each provider, along with their respective packages and pricing:

1. [Provider 1]: [Plans/Features]
2. [Provider 2]: [Plans/Features]"

"I think I prefer Provider 1 based on the features they offer."

"Great! We will now guide you through the subscription process for the selected ISP and package. Please provide your personal information to sign up, such as [Information Required]."

"Alright, I've provided my details."

"Thank you! Let me work with your ISP provider and Telemedicine provider to get you activated

"Activated. You can now access Care me App to use your service. I'm here if you need any further assistance!"

LLM Assisted User Intent Translation &
On-Demand Slice Provisioning Use case

Figure 2.3: A hypothetical interaction between an end user and an intelligent LLM, negotiating over a telemedicine service

[12]

2.3 Transformer-based Numerical Prediction in Network Traffic

This thesis offers a contrasting yet complementary perspective by focusing on predictive modeling of network traffic using Transformers adapted specifically for numerical input. Unlike traditional applications of Transformers in text processing [13], the model introduced in this thesis employs specialized embeddings for numeric features, normalization techniques for stability, and learnable positional encodings to handle time-series data. The architecture is streamlined to an encoder-only design to enhance efficiency and is tailored for continuous, multi-step forecasting. This enables the system to anticipate network demands in advance, thereby supporting more intelligent and responsive resource allocation strategies. These predictive capabilities form a core differentiation from existing LLM-based frameworks, which largely focus on interpretative and procedural automation.

2.4 Integration of Queuing Theory Models

To ground its predictions in operational reality, this research integrates well-established queuing theory models such as M/M/1, M/G/1, M/D/1, and D/D/1. These models provide a robust theoretical framework for understanding service dynamics and congestion within network slices. Prior studies such as [18] highlight the effectiveness of combining AI with theoretical models for adaptive network resource management. By combining these classical methods with predictions from a deep learning-based Transformer model, this research ensures that decisions about resource allocation are not only based on forecasts but also on proven mathematical understanding. This dual approach strengthens the system's ability to make smart, efficient choices in real-world network environments.

2.5 Limitations and Opportunities for Future Research

While this thesis significantly advances the predictive modeling of network traffic and introduces strong theoretical underpinnings, it does not yet include features such as natural language processing for user interaction or distributed agent-based coordination. These capabilities, as demonstrated by Huang et al. [11] and Dandoush et al. [12], could enhance the user-friendliness and scalability of the system.

Limitations in Related Work and How This Thesis Addresses Them:

- ChatNet (Huang et al.) lacks predictive modeling capabilities: The focus is on semantic interpretation and operational tooling, without provisions for numerical traffic forecasting. In contrast, this thesis implements Transformer-based numerical prediction [13], enabling proactive resource planning.

- ChatNet does not handle numerical time-series data: This thesis incorporates advanced handling of numerical inputs through normalization and embedding strategies, which are absent in ChatNet.

- Multi-Agent LLM Framework (Dandoush et al.) does not include traffic forecasting: The emphasis is on user intent interpretation and orchestration across domains. This thesis provides a complementary capability by introducing predictive analytics into the management cycle [17].

- Both papers lack integration with theoretical performance models: This thesis fills the gap by incorporating queuing theory [18] to guide and validate resource allocation strategies.

Future work could focus on integrating natural language interfaces and agent-based coordination mechanisms with the predictive engine to form a comprehensive, intelligent network management solution.

2.6 Summary

In conclusion, this thesis fills a crucial gap in the literature by shifting the focus from interpretative automation to predictive accuracy in network management. Through its use of adapted Transformer models [13] and queuing theory integration [18], it presents a powerful tool for dynamic resource allocation in network slicing. When the capabilities of ChatNet [11] and the multi-agent framework proposed by Dandoush et al.[12], it becomes clear that combining predictive modeling with intuitive interfaces and decentralized orchestration could yield the next generation of intelligent network management systems.

Chapter 3

Methodology

3.1 Overview

This chapter presents the methodological framework employed in this thesis, which integrates machine learning-based traffic forecasting with classical queuing theory to address the dynamic challenges in network resource allocation. The primary goal is to leverage the strengths of Transformer models [13], adapted for numerical time-series data [19], to anticipate network load conditions and make informed resource management decisions within network slicing environments. The methodology is structured around data preparation, model design, training procedures, and performance evaluation [20], providing a comprehensive blueprint for the implementation of the proposed solution.

3.2 Research Objectives

This research aims to achieve the following goals:

- Build a Transformer-based prediction model that is specifically designed to work with numerical data from network traffic [21].
- Compare how well this model performs against more traditional methods like LSTM [22] and simple neural networks [23].
- Use the predictions from the model together with queuing theory models (M/M/1,

M/G/1, D/D/1, and M/D/1) [24] to help plan and manage network resources in real time.

- Measure how accurate and useful the system is, including how well it uses resources and maintains service quality [25].
- Demonstrate how this integrated system can support intelligent decision-making in future 5G/6G network environments [26].

3.3 Data Collection and Preprocessing

The dataset used in this research comprises historical network traffic records, including parameters such as bandwidth usage, packet rates, and service requests [27]. These records are typically collected at regular intervals, forming a time-series dataset.

Data preprocessing involves the following steps:

- **Data Cleaning:** Removal of outliers, handling of missing values, and filtering of irrelevant data points [28].
- **Normalization:** Application of min-max scaling or standardization to ensure feature values are within a consistent range and to stabilize training [29].
- **Time Encoding:** Incorporation of temporal context using learnable positional embeddings [13], which allows the model to understand sequence ordering and temporal dependencies.

3.4 Transformer Model Architecture

The core of the forecasting system is an encoder-only Transformer architecture adapted for numerical data [19]. Key components include:

- **Numerical Embedding Layer:** Converts continuous numerical features into vector representations suitable for input into the Transformer [30].

- **Multi-Head Attention:** Enables the model to attend to different positions in the sequence simultaneously, capturing complex temporal relationships [13].
- **Feedforward Network:** Applies non-linear transformations to the attention outputs, enhancing model expressiveness [20].
- **Positional Encoding:** Uses learnable embeddings to represent the position of each time step, ensuring the model understands sequence order [13].

3.5 Model Training and Validation

The Transformer model is trained using supervised learning, where historical data sequences are mapped to future values [31]. Key training aspects include:

- **Loss Function:** Mean Squared Error (MSE) is used to penalize the difference between predicted and actual values [23].
- **Data Split:** The dataset is divided into training (70), validation (15), and test (15) sets using chronological splitting to maintain temporal integrity [32].
- **Hyperparameter Optimization:** Grid search or random search is used to find optimal values for learning rate, number of layers, and attention heads [33].

3.6 Integration with Queuing Models

Predicted traffic patterns are fed into multiple queuing models to evaluate and manage network load. The models used in this thesis include:

- **M/M/1 Model:** Assumes both arrival and service times follow exponential distributions. Useful for systems with memoryless behavior [24].
- **M/G/1 Model:** Assumes exponentially distributed arrivals with general service time distributions, providing flexibility for varied service characteristics [34].

- **D/D/1 Model:** Assumes deterministic arrival and service times. Ideal for analyzing systems with highly predictable traffic patterns [35].
- **M/D/1 Model:** Combines exponential arrivals with deterministic service times, reflecting situations where service mechanisms are uniform but arrivals are random [24].

3.7 Evaluation Metrics

The proposed system is evaluated using the following metrics:

- **Prediction Metrics:** Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2) [36].
- **Queuing Metrics:** Average wait time, server utilization, and queue length [34].
- **Comparative Analysis:** Model performance is benchmarked against LSTM and neural regression to validate improvements [22, 23].

3.8 Implementation Tools and Environment

The implementation is carried out using Python, leveraging libraries such as PyTorch or TensorFlow for deep learning [37, 38], and SciPy or custom scripts for queuing model simulation [39]. Experiments are conducted on GPU-enabled machines to accelerate training.

3.9 Summary

This methodology integrates state-of-the-art deep learning techniques with classical queuing theory to build an intelligent, predictive framework for network slicing management. It addresses existing limitations in current research by offering both predictive insight and theoretical rigor, enabling more responsive and efficient network resource allocation [25].

Chapter 4

Model Implementation

4.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are a class of neural networks designed to model sequential data by maintaining a hidden state that captures information from previous time steps [22]. This property makes RNNs suitable for time-series and sequence prediction tasks.

At each time step t, the hidden state $\mathbf{h}_t$ is updated based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$:

$$\mathbf{h}_t = \sigma_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{4.1}$$

where $\mathbf{W}_{xh}$ and $\mathbf{W}_{hh}$ are weight matrices, $\mathbf{b}_h$ is the bias vector, and σ_h is the activation function (usually tanh or ReLU).

The output $\mathbf{y}_t$ is computed as:

$$\mathbf{y}_t = \sigma_y(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y) \tag{4.2}$$

Here, $\mathbf{W}_{hy}$ and $\mathbf{b}_y$ are weights and bias for the output layer, and σ_y is an activation function.

Limitations:

- Traditional RNNs suffer from the vanishing and exploding gradient problems [22], which hinder their ability to learn long-term dependencies in sequences.
- Due to their limited memory, RNNs often forget information from earlier time steps in long sequences.
- LSTM and Transformer models are better suited for capturing long-term dependencies through memory cells and attention mechanisms, respectively [13].
- Transformers support parallel processing across all time steps, which significantly speeds up training compared to RNNs [13].

.

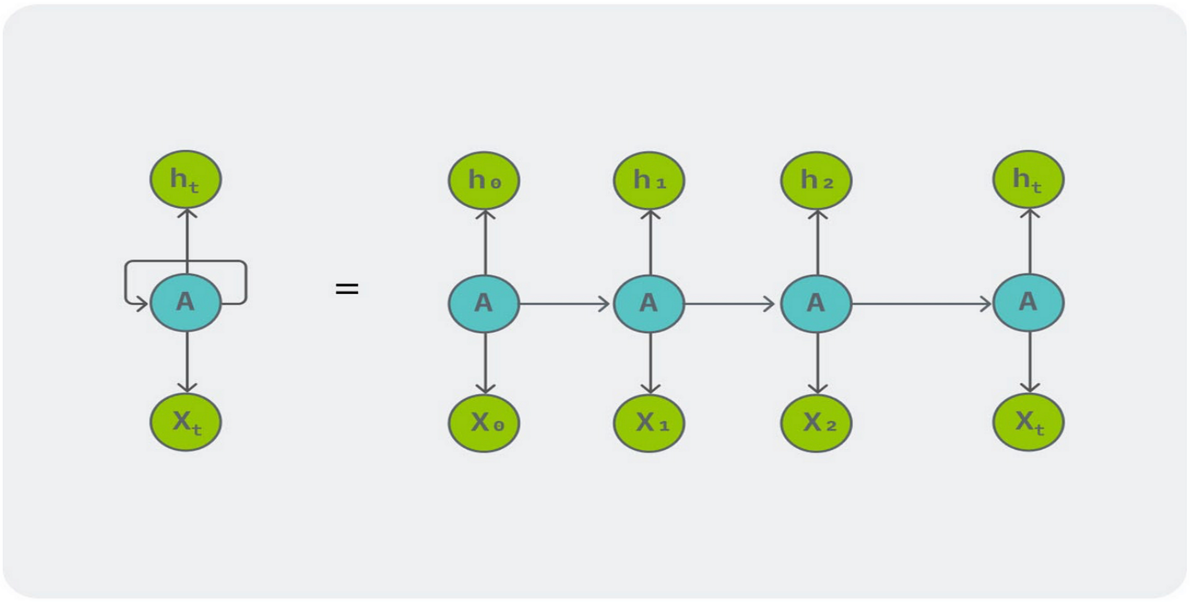


Figure 4.1: Basic Recurrent Neural Network architecture

4.2 Long Short-Term Memory (LSTM)

LSTM networks are a special type of RNN designed to overcome the vanishing gradient problem by introducing memory cells and gating mechanisms that control the flow of information [22].

An LSTM cell has three main gates: input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, and output gate $\mathbf{o}_t$. The cell state is denoted as $\mathbf{c}_t$.

The equations governing LSTM at time step t are:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{4.3}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{4.4}$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \tag{4.5}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{4.6}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{4.7}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{4.8}$$

where: - σ is the sigmoid activation function, - $\odot$ denotes element-wise multiplication, - $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_c, \mathbf{W}_o$ are weight matrices, - $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_c, \mathbf{b}_o$ are biases.

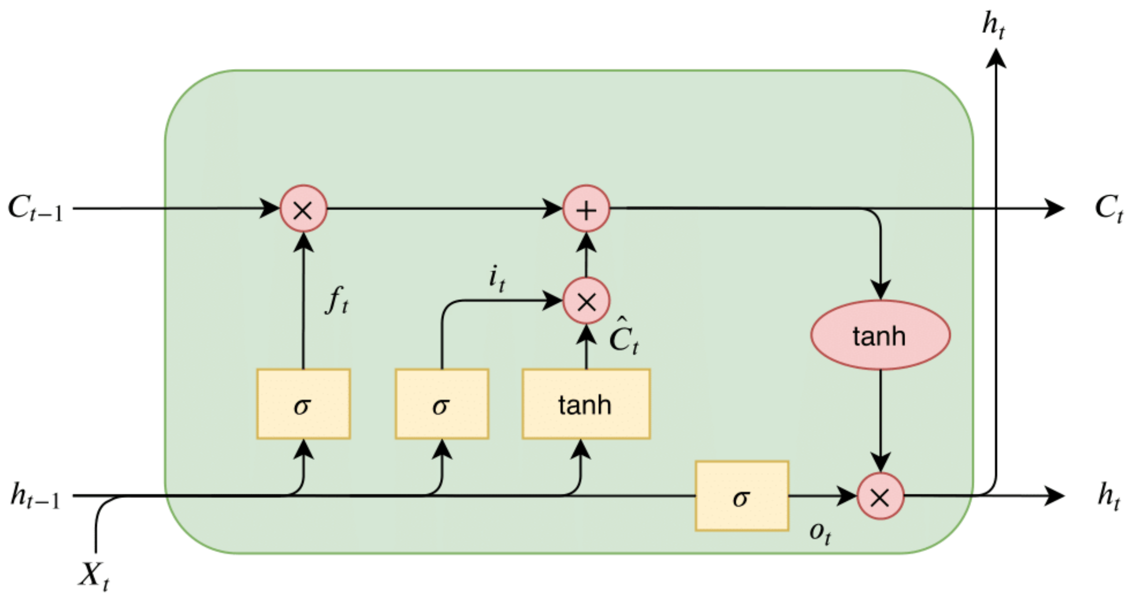


Figure 4.2: Structure of an LSTM with gates

4.3 Transformer Model

Transformers rely on self-attention mechanisms to model dependencies between input elements regardless of their distance in the sequence, allowing better parallelization compared to RNNs [13].

4.3.1 Self-Attention Mechanism

Given an input sequence represented by matrix $\mathbf{X}$, the self-attention mechanism computes three matrices : Query $\mathbf{Q}$, Key $\mathbf{K}$, and Value $\mathbf{V}$ as [13]:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V \tag{4.9}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ are learned parameter matrices.

The scaled dot-product attention is computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{4.10}$$

where d_k is the dimension of the key vectors, used for scaling.

4.3.2 Multi-Head Attention

To allow the model to attend to information from different representation subspaces, multi-head attention concatenates several self-attention outputs [13]:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, ..., \text{head}_h)\mathbf{W}^O \tag{4.11}$$

where each head is:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{4.12}$$

4.3.3 Positional Encoding

Since Transformers have no recurrence, positional information is added to the input embeddings [13]:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.13}$$

where *pos* is the position and *i* is the dimension.

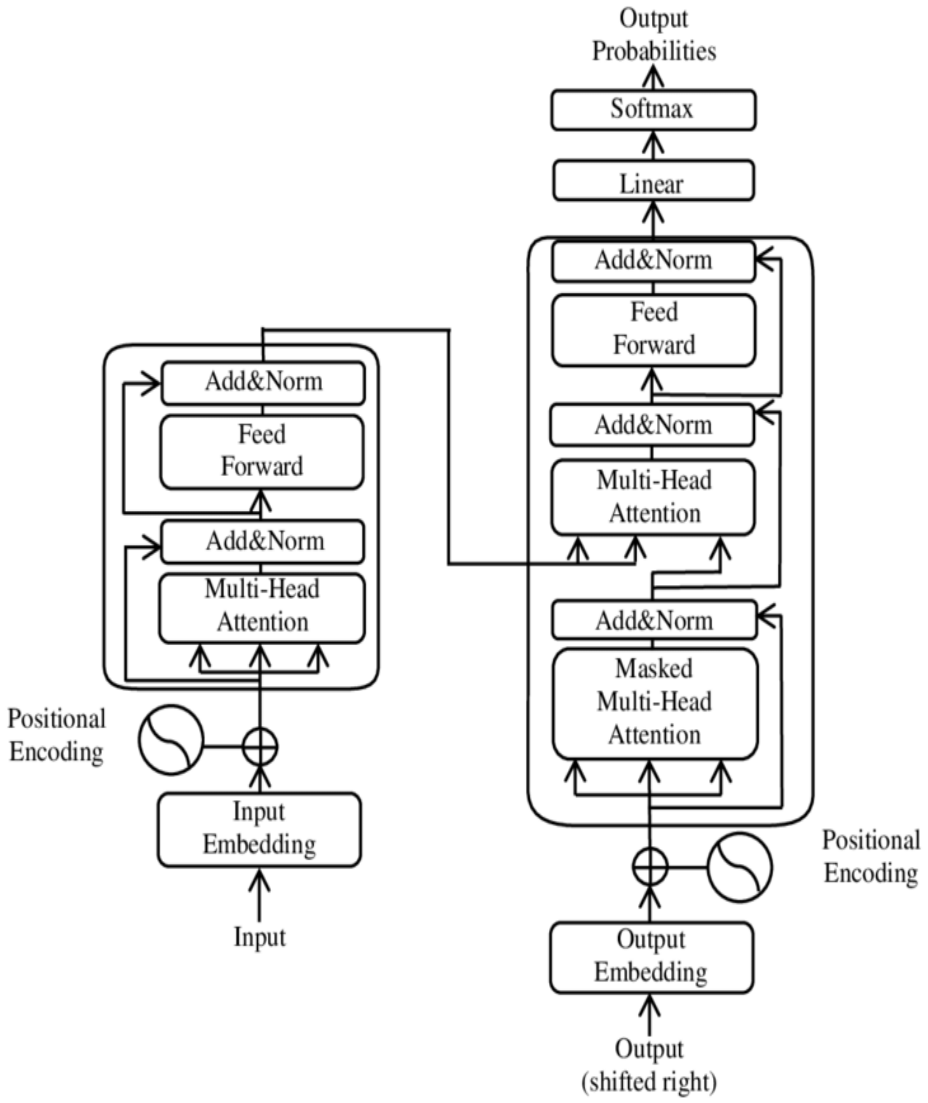


Figure 4.3: Transformer architecture overview

[13]

4.4 Summary

In this chapter, we described the key deep learning models used for network traffic prediction: RNN, LSTM, and Transformer. The limitations of vanilla RNNs motivated the use of LSTM networks with gating mechanisms. Further, the Transformer architecture leverages self-attention for capturing long-range dependencies efficiently [13]. These models were implemented using Python frameworks in our experiments [37, 38].

4.5 Modified Transformer Model

The core contribution of this thesis is the successful adaptation of the Transformer model originally designed for natural language processing (NLP) tasks with textual input to forecast numerical time series data in the context of network slicing in 5G/6G networks. This required several architectural modifications to enable the Transformer to handle the structure and dynamics of time series input, which is fundamentally different from text based input.

4.5.1 Motivation

The Transformer model has shown excellent performance in modeling long-range dependencies in NLP tasks. However, its standard architecture assumes tokenized text sequences as input, which is not directly suitable for time-dependent numerical data like network traffic measurements. To leverage the strength of the Transformer for time series forecasting, this thesis introduces a series of modifications to the original model.

4.5.2 Proposed Modifications to the Transformer Model

1. **Input Representation:**
   - Replaced word embeddings with embeddings designed for numerical time series features.

- Applied normalization techniques such as Min-Max Scaling and Standardization to stabilize training.

2. **Positional Encoding:**
    - Replaced fixed sinusoidal positional encodings with learnable positional embeddings.
    - This allows the model to learn the temporal structure of the data more flexibly.

3. **Architecture Simplification:**
    - Removed the decoder component of the original Transformer.
    - Used only the encoder for direct regression tasks, which simplifies the model and reduces computational cost.

4. **Output Layer:**
    - Replaced the classification layer (used in NLP) with a regression output layer.
    - Enabled the model to produce continuous-valued outputs for multi-step forecasting.

4.5.3 Significance of the Contribution

These modifications enable the Transformer to process and forecast network traffic patterns effectively. Unlike traditional recurrent models such as LSTM and GRU, the modified Transformer captures long-range dependencies more efficiently and provides improved forecasting accuracy for dynamic network slicing scenarios.

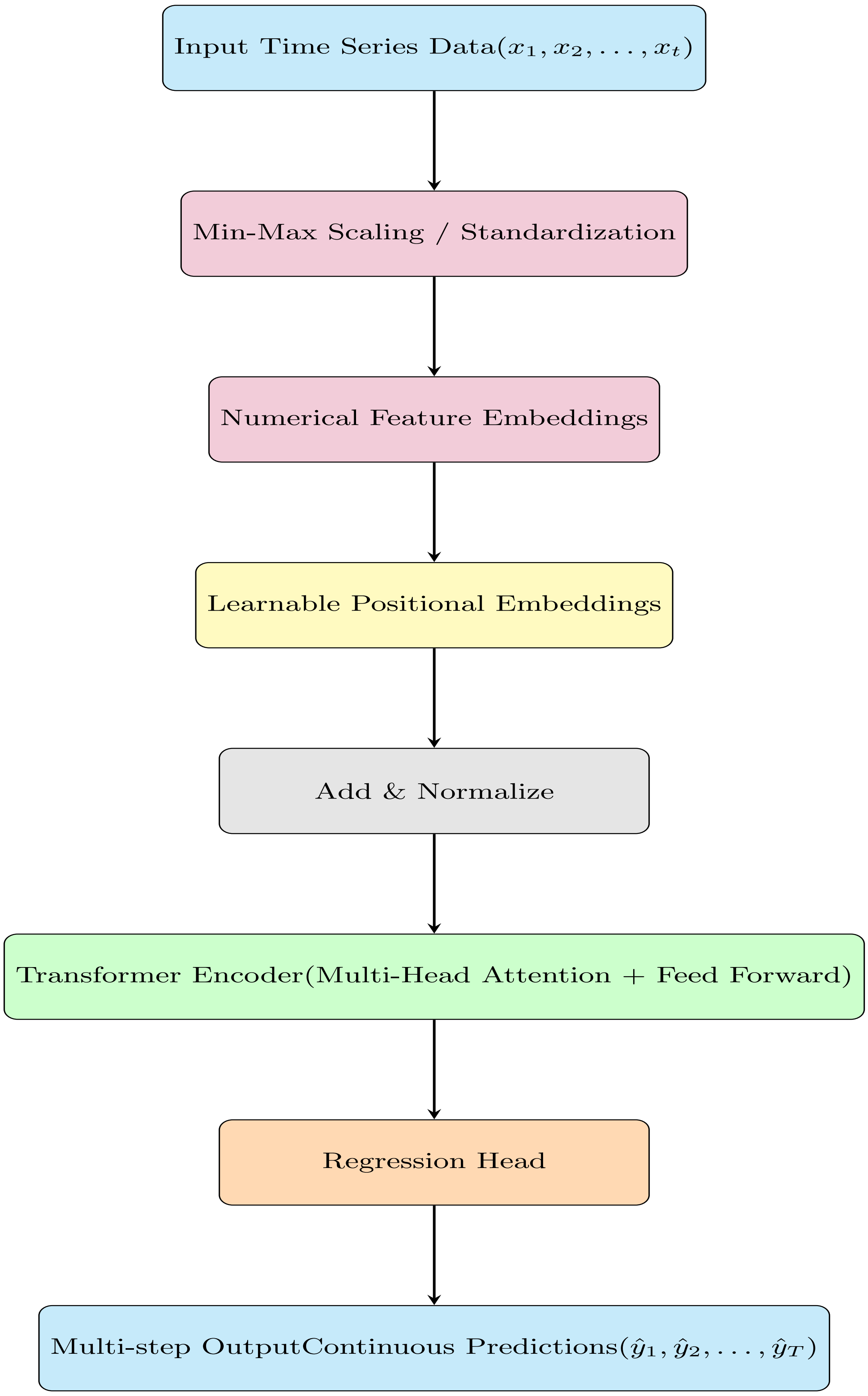

Input Time Series Data$(x_1, x_2, \ldots, x_t)$
Min-Max Scaling / Standardization
Numerical Feature Embeddings
Learnable Positional Embeddings
Add & Normalize
Transformer Encoder(Multi-Head Attention + Feed Forward)
Regression Head
Multi-step OutputContinuous Predictions$(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T)$

Figure 4.4: Transformer architecture overview

4.5.4 Comparison with Standard Transformer

Table 4.1: **Comparison between Standard and Modified Transformer**

Aspect	**Standard Transformer**	**Modified Transformer**
Input Type	Word Embeddings (Text)	Numerical Feature Embeddings
Positional Encoding	Fixed Sinusoidal	Learnable Positional Embeddings
Architecture	Encoder + Decoder	Encoder Only
Output Type	Token Probabilities	Continuous Numerical Values
Application Domain	NLP Tasks	Time Series Forecasting in Network Traffic

4.5.5 Conclusion

The successful adaptation of the Transformer model for numerical time series forecasting represents a contribution to the field of deep learning for network slicing. It demonstrates that with appropriate modifications, attention-based architectures can outperform traditional sequence models in tasks involving complex temporal patterns.

Chapter 5

Data Collection and Preprocessing

5.1 Introduction

Network prediction tasks rely heavily on high-quality time-series data that accurately reflects real-world communication behavior [27]. This chapter describes the theoretical background of queueing models used to simulate such network behavior and the preprocessing steps performed to prepare the data for LSTM and Transformer-based prediction models [22, 13].

We simulate five classical queueing models: M/M/1, D/D/1, M/G/1, and M/D/1 [24]. These models help generate synthetic data representing network metrics such as arrival time, service time, latency, and jitter. After simulation, the dataset is normalized and organized into input-output sequences suitable for model training.

5.2 Theoretical Background of Queueing Models

Queueing theory studies the behavior of queues systems where entities (e.g., data packets) wait for service. Each model is defined by its arrival and service time distribution [34]. The general format of queueing notation is A/B/1, where:

- A: Arrival time distribution
- S: Service time distribution

- 1: Number of servers

5.2.1 M/M/1 Queue (Exponential/Exponential/1)

The M/M/1 queueing model is one of the fundamental models in queueing theory, known for its simplicity and analytical tractability [24, 34]. It represents a system where both arrivals and service times are governed by memoryless (Markovian) processes.

- **Arrival Process:** Customers or data packets arrive according to a Poisson process with an average rate of λ arrivals per unit time.
- **Service Process:** Each service time is exponentially distributed with an average service rate of μ customers per unit time.
- **Number of Servers:** A single server processes incoming requests in the order they arrive (First-Come, First-Served).

Assumptions:

- The queue has an infinite capacity, so no arrivals are lost.
- The arrival and service processes are independent.
- The system is stable only if $\lambda < \mu$, ensuring the queue does not grow indefinitely.

Performance Metrics:

- **Utilization Factor (Traffic Intensity):**

$$\rho = \frac{\lambda}{\mu}$$

  This indicates the fraction of time the server is busy. Stability requires $\rho < 1$.

- **Average Number of Packets in System (L):**

$$L = \frac{\rho}{1-\rho}$$

Represents the expected total number of customers in the system (waiting + being served).

- **Average Time in System (W):**

$$W = \frac{1}{\mu - \lambda}$$

The average time a customer spends in the system, including both queueing and service time.

- **Average Time in Queue (W˙q):**

$$W_q = \frac{\lambda}{\mu(\mu - \lambda)} = \frac{\rho}{\mu - \lambda}$$

The expected waiting time before service begins.

- **Average Number of Packets in Queue (L˙q):**

$$L_q = \lambda W_q = \frac{\lambda^2}{\mu(\mu - \lambda)}$$

The average number of customers waiting in line (excluding the one being served).

Use Cases:

- Modeling customer service desks or call centers with random arrival and service times.
- Useful in networking to analyze buffer behavior in routers and switches under random packet arrival.

5.2.2 D/D/1 Queue (Deterministic/Deterministic/1)

The D/D/1 queue is a basic and idealized model in queueing theory. It represents a system where both the inter-arrival times and service times are strictly deterministic and occur at constant, fixed intervals [35].

- **Arrival Process:** Jobs or packets arrive exactly every $\frac{1}{\lambda}$ units of time, where λ is the fixed arrival rate.
- **Service Process:** Each job requires a fixed service time of $\frac{1}{\mu}$, where μ is the constant service rate.
- **Number of Servers:** A single server handles incoming jobs on a First-Come, First-Served (FCFS) basis.

Key Features:

- There is no randomness in the system.
- Since both arrivals and service times are perfectly regular, the behavior of the queue is highly predictable.

Performance Characteristics:

- **Utilization:**

$$\rho = \frac{\lambda}{\mu}$$

  Indicates how busy the server is. For a stable system, $\rho \leq 1$. If $\rho > 1$, the queue will grow without bound.

- **Queue Behavior:**
  - If $\lambda < \mu$, the server will have idle time between jobs.
  - If $\lambda = \mu$, the system reaches a balanced state with no queue buildup.
  - If $\lambda > \mu$, jobs accumulate and the queue length increases linearly.

- **Waiting Time:** In the stable case, when $\lambda \leq \mu$, jobs do not wait in the queue because each arrives just in time to be served. Thus:

$$W_q = 0, \quad W = \frac{1}{\mu}$$

- **Queue Length:** In the stable condition, the queue length remains zero or bounded by one depending on the arrival-service timing alignment.

Applications:

- Systems with highly predictable traffic, such as industrial automation or real-time control systems.
- Used for benchmarking other queueing models due to its ideal and noise-free behavior.

5.2.3 M/G/1 Queue (Markovian/General/1)

The M/G/1 queue is a fundamental single-server queueing model where the arrival process is stochastic (Poisson), but the service times follow a general (arbitrary) distribution. It provides a more realistic representation of many real-world systems where service times are not necessarily exponential [24, 34].

- **Arrival Process:** Poisson process with arrival rate λ (exponentially distributed inter-arrival times).
- **Service Process:** General distribution with mean service time $E[S]$ and variance $\text{Var}(S)$.
- **Number of Servers:** One server (single-server system).

Key Characteristics:

- The arrival pattern is memoryless, but the service time can follow any distribution (e.g., uniform, normal, or heavy-tailed).
- The model is useful for analyzing systems with irregular service durations such as file transfers, machine repairs, or processing times.

- System behavior is captured using the Pollaczek–Khinchine (P-K) formula, which provides expressions for performance metrics.

Important Metrics:

- **Traffic Intensity (Utilization):**

$$\rho = \lambda \cdot E[S]$$

  For the system to be stable, $\rho < 1$.

- **Average Number of Packets in System:**

$$L = \rho + \frac{\lambda^2 \cdot \mathrm{Var}(S)}{2(1-\rho)}$$

- **Average Waiting Time in Queue:**

$$W_q = \frac{\lambda \cdot \mathrm{Var}(S)}{2(1-\rho)}$$

- **Average Time in System (Waiting + Service):**

$$W = W_q + E[S]$$

- **Average Queue Length:**

$$L_q = \lambda \cdot W_q$$

Explanation:

- The variance of the service time plays a critical role in system performance.
- Higher variability in service times (higher $\mathrm{Var}(S)$) leads to longer waiting times and larger

5.2.4 M/D/1 Queue (Markovian/Deterministic/1)

- **Arrival Process:** Follows a Poisson distribution with exponential inter-arrival times (memoryless property).
- **Service Process:** Each job or packet has a fixed (deterministic) service time.
- **Number of Servers:** A single server is available to serve incoming requests.

In this queueing model, variability is introduced only by the arrival process, as service times are constant. This makes the analysis simpler than in more general models [35].

The average waiting time in the queue W_q for the M/D/1 system is given by:

$$W_q = \frac{\rho^2}{2\mu(1-\rho)}$$

Where:

- $\rho = \lambda/\mu$ is the traffic intensity.
- λ: Mean arrival rate.
- μ: Mean service rate, which is the inverse of the constant service time.

Key Features:

- The deterministic nature of service time reduces variability in system behavior.
- As compared to M/M/1, the M/D/1 model yields shorter average waiting times because there is no randomness in service.
- Useful for systems where processing time is known and consistent, such as embedded systems or scheduled tasks in computing.

5.3 Network Metrics Captured

During the simulation of traffic and queuing models, various performance metrics were recorded at each time step. These metrics provide insight into the system's behavior and are essential for evaluating model performance [25].

5.3.1 Arrival Time

The arrival time refers to the exact moment when a packet or data unit enters the queueing system. This metric helps in calculating inter-arrival times and tracking system load over time.

5.3.2 Service Time

Service time is the duration taken by the server to process a packet. It depends on the server's speed and the complexity or size of the incoming request. The value is often drawn from a predefined distribution (e.g., constant, exponential, or general).

5.3.3 Latency

Latency represents the total delay experienced by a packet from the moment it arrives until it exits the system after being serviced. It is computed as:

$$\text{Latency} = \text{Waiting Time} + \text{Service Time}$$

High latency may indicate congestion or limited server capacity and is a critical metric in evaluating Quality of Service (QoS).

5.3.4 Jitter

Jitter quantifies the variation in latency between consecutive packets. In real-time systems such as voice or video communication, low jitter is crucial. It is defined as:

$$\text{Jitter}_t = \left|\text{Latency}_t - \text{Latency}_{t-1}\right|$$

Higher jitter values indicate inconsistency in system response time, which can degrade the performance of time-sensitive applications.

5.3.5 Queue Length

This metric measures the number of packets waiting in the queue at any given time step. It reflects the current load on the system and helps in identifying periods of congestion. A consistently long queue may suggest under-provisioned resources or inefficient scheduling.

5.4 Data Preprocessing

To ensure that the collected simulation data is suitable for training deep learning models, several preprocessing steps were applied.

5.4.1 Handling Missing Values

Although simulations are controlled, some generated datasets may contain missing or undefined values due to abrupt termination or simulation errors. These gaps were handled using [28]:

- **Forward Fill:** Propagates the last valid observation forward.
- **Interpolation:** Estimates missing values using trends in neighboring data points.

These techniques help maintain continuity in the time series without introducing bias.

5.4.2 Feature Normalization

Normalization is essential to scale all numerical features to a common range, typically [0, 1], which helps improve training stability and convergence speed. Min-max scaling was applied as follows:

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

This ensures that large-scale features do not dominate smaller ones during model training.

5.4.3 Sequence Preparation

Since LSTM and Transformer models operate on sequential data, the time-series metrics were transformed into a supervised learning format using a sliding window technique. This involved:

- **Input Sequence:** A window of length n capturing values from time t to $t + n - 1$.
- **Target Value:** The immediate next value at time $t + n$, which the model is trained to predict.

This method allows the model to learn temporal dependencies and trends across the dataset.

5.4.4 Train-Test Splitting

To evaluate model performance reliably, the dataset was divided as follows:

- **Training Set:** 80% of the dataset used to learn model parameters [32].

  – **Testing Set:** 20% of the dataset used for evaluating generalization on unseen data.

Importantly, shuffling was avoided to preserve the chronological order of events, which is vital in time-series prediction tasks.

5.5 Summary

In this chapter, we detailed the essential queueing metrics captured during simulation, such as arrival time, service time, latency, jitter, and queue length. These metrics provide a comprehensive view of system behavior under different traffic conditions. We also described how the raw data was preprocessed to make it suitable for training deep learning models. This included missing value handling, feature scaling, temporal sequence generation, and careful dataset partitioning. The resulting structured dataset enables the effective application of LSTM and Transformer models in predicting network traffic and assessing queuing performance in dynamic network slicing environments.

Chapter 6

Experimental Setup

6.1 Introduction

This chapter provides a detailed explanation of the experimental environment used for evaluating the proposed network prediction models. The chapter outlines the hardware and software platforms, simulation techniques, model architectures, hyperparameters used in training, and the metrics used to assess prediction performance. The goal is to ensure reproducibility and clarity of the conducted experiments.

6.2 Hardware and Software Environment

6.2.1 Hardware Configuration

All simulations, model training, and evaluations were conducted on a personal computing system equipped to handle moderate machine learning workloads. The configuration of the hardware setup is as follows:

- **Processor:** Intel Core i7-11800H with 8 cores and 16 threads, operating at a base frequency of 2.30 GHz.
- **Memory (RAM):** 16 GB DDR4, offering sufficient capacity for in-memory computations during training and data preprocessing.

– **Graphics Processing Unit (GPU):** NVIDIA GeForce GTX 1660 Ti with 6 GB of dedicated VRAM, enabling hardware acceleration for deep learning tasks.

– **Storage:** 512 GB NVMe SSD, providing high-speed data read/write operations to improve dataset loading and model checkpointing efficiency.

6.2.2 Software Tools

The entire implementation pipeline, from data generation to model training and evaluation, was developed using Python 3.10. Various open-source libraries were employed for specific tasks throughout the project [39, 38, 37]:

– **NumPy:** Used for numerical operations such as array manipulation, mathematical computations, and random number generation.

– **Pandas:** Utilized for structured data processing, including data cleaning, transformation, and tabular representation.

– **Matplotlib & Seaborn:** Applied for generating detailed plots and visualizations to analyze trends in queue metrics and model predictions.

– **Scikit-learn:** Employed for data preprocessing steps like normalization and for computing evaluation metrics such as RMSE and MAE.

– **TensorFlow & Keras:** Used for implementing and training the LSTM-based models, taking advantage of built-in layers and optimizers.

– **PyTorch:** Adopted for the Transformer model implementation, offering a flexible and dynamic computation graph suitable for experimentation.

– **Google Colab & Visual Studio Code:** Served as the main development environments for writing, debugging, and running code.

6.3 Dataset Generation and Configuration

6.3.1 Simulated Queueing Models

The dataset was synthetically generated through the simulation of five widely studied queueing models: M/M/1, D/D/1, M/G/1 and M/D/1. Each model represents a different configuration of arrival and service time distributions, simulating a variety of real-world traffic scenarios. These simulations were implemented using Python scripts and involved tracking various network performance metrics over time [24, 34].

For each run of the simulation, packets were generated based on the arrival distribution, and their service times were assigned according to the specific model. The system then recorded key performance indicators including waiting time in the queue, total latency, and packet-to-packet jitter.

6.3.2 Dataset Structure

The final dataset used for training and evaluation had the following properties:

- **Features Captured:** Arrival Time, Service Time, Waiting Time, Latency, and Jitter.
- **Total Samples:** 50,000 time steps were simulated, providing a diverse and statistically meaningful set of data points.
- **Sequence Length:** Each input sample consisted of a sequence of 10 consecutive time steps (i.e., past 10 observations).
- **Prediction Objective:** The target variable was the corresponding value of a selected metric at the next (11th) time step.
- **Train-Test Division:** 80% of the data was allocated for training, and the remaining 20% was used for testing.

6.3.3 Preprocessing Steps

Before model training, the dataset underwent several preprocessing steps to ensure consistency, numerical stability, and compatibility with sequential learning architectures:

- **Handling Missing Values:** Although rare in simulation-generated data, any missing entries were addressed using forward-fill or linear interpolation methods to maintain sequence continuity [28].
- **Normalization:** All features were scaled using min-max normalization to bring values into the [0, 1] range, reducing the risk of gradient explosion or vanishing during training [29].
- **Sequence Construction:** A sliding window of size 10 was applied across the time series to form overlapping input sequences, with the 11th step used as the prediction target [31].
- **Temporal Integrity:** The data was kept in its natural order without shuffling, to preserve time dependencies essential for sequence-based models like LSTM and Transformer.

6.4 Model Configurations

6.4.1 LSTM Model Details

A deep learning model based on the Long Short-Term Memory (LSTM) architecture was designed to predict future values of queueing metrics based on historical sequences [22]. The model architecture is described below:

- **Input Layer:** Accepts input tensors of shape (10, 5), where 10 is the time window and 5 is the number of features.

- **LSTM Layer:** A single LSTM layer with 64 memory units and tanh activation was used to capture temporal patterns and dependencies.
- **Dropout Layer:** A dropout rate of 0.2 was applied after the LSTM layer to mitigate overfitting by randomly disabling neurons during training.
- **Dense Layer:** A fully connected layer with 32 units and ReLU activation was included to transform the extracted temporal features.
- **Output Layer:** A single neuron with linear activation was used to predict the target value (e.g., future latency or jitter).

This configuration provides a balance between model complexity and performance, making it suitable for medium-sized datasets like the one used in this study. The model was compiled using the Adam optimizer and Mean Squared Error (MSE) as the loss function.

6.4.2 Modified Transformer Model for Time Series Forecasting in Network Traffic

To effectively apply the Transformer architecture for time series forecasting in a network slicing context, several modifications were made to adapt it for handling numerical data and direct regression tasks [13].

Input Representation (Handling Numerical Data)

Unlike the traditional Transformer, which uses word embeddings for text input, our model processes multivariate numerical time series data. Each input vector at a time step includes normalized values of five features: Arrival Time, Service Time, Waiting Time, Latency, and Jitter. To bring the input values within a comparable range and stabilize training, Min-Max Scaling or Standardization was applied. Each

5-dimensional vector is projected into a 64-dimensional embedding space using a linear transformation layer.

Positional Encoding (Incorporating Temporal Order)

Transformers do not have a built-in mechanism for representing the sequence order. To encode temporal information, we used learnable positional embeddings rather than the traditional fixed sinusoidal encodings. These learnable embeddings allow the model to adapt positional patterns specific to network traffic data over time, improving prediction performance.

Architecture Adjustments

Instead of using the full encoder-decoder architecture, we implemented only the encoder component. This is sufficient for time series regression, as the task involves predicting future values based on past observations rather than generating sequences or translations. The encoder stack includes:

- Two Transformer encoder layers
- Four multi-head self-attention mechanisms per layer
- Residual connections and layer normalization
- Position-wise feedforward networks with 128 hidden units and ReLU activation

Output Layer for Multi-Step Regression

The encoded sequence is reduced via pooling (e.g., mean pooling across time steps), producing a fixed-size context vector. This is passed through a fully connected dense layer followed by a linear output layer that predicts a continuous numerical value. For multi-step forecasting, the model can be extended to output multiple values simultaneously or be used in an autoregressive fashion.

Model Performance and Advantages

The modified Transformer demonstrated superior performance compared to the LSTM model in capturing long-range dependencies within network slicing scenarios. Key benefits included:

- Parallel processing of sequence data, leading to faster training times
- Better handling of temporal variability and irregular traffic patterns
- Improved accuracy in long-term forecasting due to global attention mechanisms

These enhancements make the Transformer particularly effective for dynamic network resource prediction, which is critical in modern 5G and 6G network slicing environments.

6.5 Training Setup

To train the proposed LSTM and Transformer-based forecasting models, a consistent and optimized training configuration was used to ensure fair comparison and reliable convergence. The key parameters for the training process are outlined below:

- **Optimizer:** The Adam optimizer was used due to its adaptive learning rate capabilities and efficient handling of sparse gradients [?]. It combines the benefits of both AdaGrad and RMSProp.
- **Loss Function:** Mean Squared Error (MSE) was selected as the primary loss function, suitable for continuous regression tasks where minimizing prediction error is essential.
- **Batch Size:** A mini-batch size of 64 was chosen to strike a balance between training speed and convergence stability.

- **Epochs:** Each model was trained for a maximum of 300 epochs, which was sufficient to allow convergence for both LSTM and Transformer architectures.
- **Learning Rate:** The initial learning rate was set to 0.001. This value was empirically chosen to ensure smooth optimization without overshooting minima.
- **Early Stopping:** To prevent overfitting, early stopping was enabled with a patience parameter of 10 epochs. Training was halted if the validation loss did not improve for 10 consecutive epochs.

This training setup was consistent across all experiments, ensuring that any observed differences in model performance were due to model architecture rather than differing hyperparameters.

6.6 Evaluation Metrics

To assess the forecasting accuracy and performance of the models, multiple error metrics were employed. These metrics provide a comprehensive view of prediction quality from different perspectives [36]:

- **Mean Squared Error (MSE):** Measures the average of the squared differences between predicted and actual values. It is sensitive to large errors.

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

- **Mean Absolute Error (MAE):** Computes the average absolute difference between the predicted and true values. It treats all errors equally.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Root Mean Squared Error (RMSE):** The square root of MSE, which restores the error unit to the original scale of the data. It penalizes larger errors more severely.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- **Coefficient of Determination (R^2 Score):** Indicates how well the model explains the variability of the target values. An R^2 value close to 1 indicates strong predictive performance.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

These metrics were calculated on the test dataset after training and were used to evaluate both single-step and multi-step forecasting performance.

6.7 Summary

This chapter presented a detailed overview of the experimental setup used in this research work. It began with the hardware and software environment, highlighting the tools and computational resources employed. Then, it described the simulation-based dataset generation process using various queuing models such as M/M/1, D/D/1, M/G/1, M/D/1. Key features such as arrival time, latency, service time, and jitter were extracted and preprocessed using normalization and sequence formatting techniques.

The chapter also elaborated on the architectural configurations of the LSTM and Transformer models [22, 13]. Specific adaptations were made to the Transformer to make it suitable for time series regression. Furthermore, the training setup was carefully designed with standardized hyperparameters and regularization strategies such as early stopping.

Finally, appropriate evaluation metrics such as MSE, MAE, RMSE, and R^2 were defined to quantitatively measure the forecasting accuracy of the models. This well-structured foundation sets the stage for the result analysis and interpretation discussed in the following chapter.

Chapter 7

Results And Analysis

7.1 Visualization of Results

This section provides visual insights into the prediction quality of the deep learning models across different queueing systems. Each model exhibits unique arrival and service patterns, and the following figures demonstrate how well the models learn and predict these patterns [31].

7.1.1 D/D/1 Queue Model

The D/D/1 model represents a fully deterministic queue where both arrival and service times are constant. It's the simplest system, often used as a theoretical benchmark.

Key Features:

- The model accurately learns fixed intervals in both arrivals and service.
- Small deviations may appear due to limited floating-point precision or training noise.
- The latency remains constant, showing the Transformer's effectiveness in modeling deterministic queuing systems.

Conclusion: The D/D/1 queue allows the Transformer model to demonstrate its learning consistency. Although the system lacks variability, the model successfully aligns with the fixed patterns, validating its reliability for modeling deterministic traffic flows .

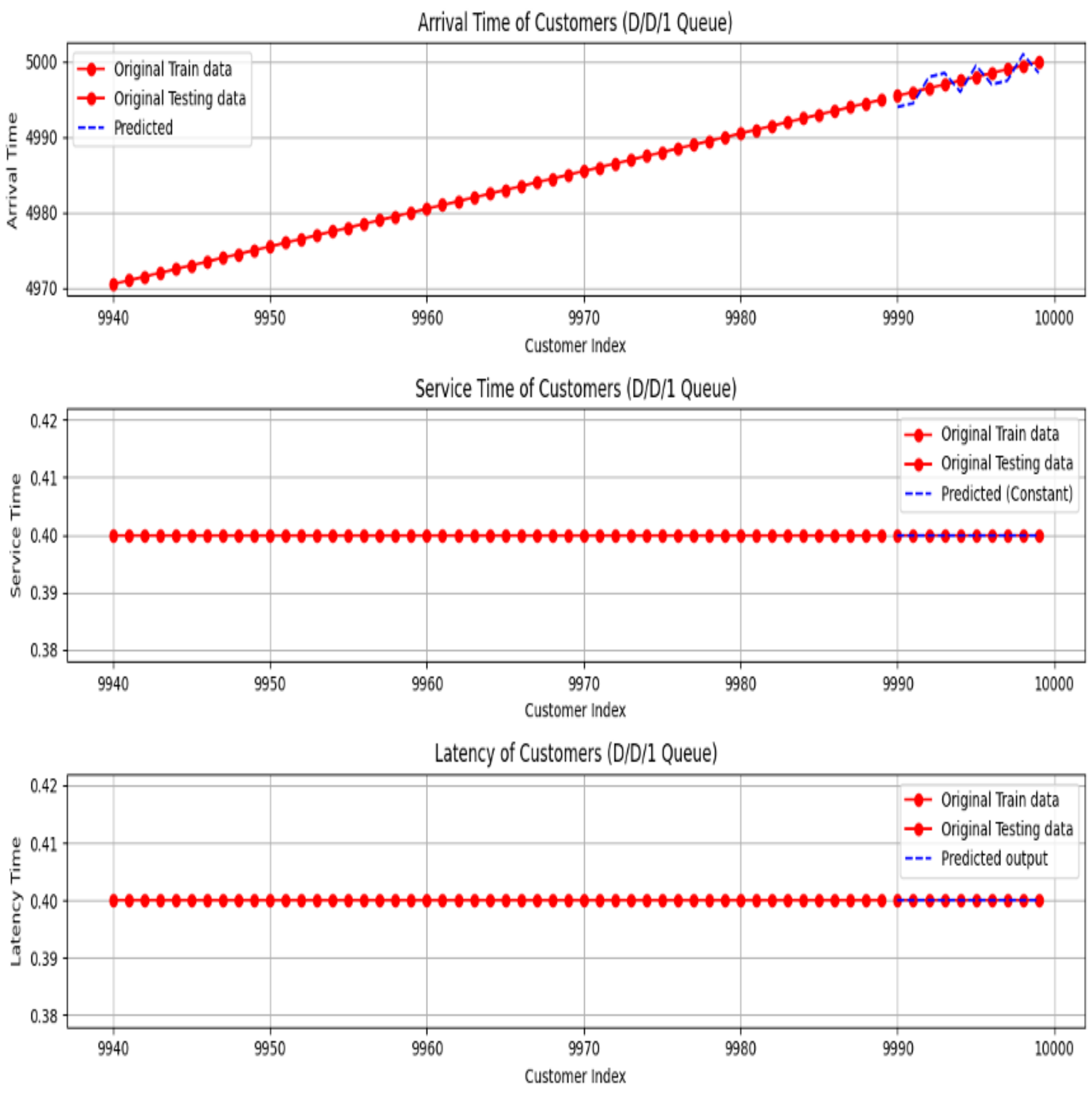


Figure 7.1: Transformer Model Predictions vs Actual for D/D/1 Queue

As shown in Figure 7.1, Transformer models achieve high accuracy, reflecting the model's ease of learning in deterministic settings.

7.1.2 M/G/1 Queue Model

The M/G/1 model assumes a Markovian (Poisson) arrival process and a general service time distribution. It introduces service time variability while keeping arrivals random but memoryless.

Key Features:

- The Transformer captures the randomness in arrival patterns (Poisson-like) .
- It effectively approximates the general service time distribution.
- Predicted latency values closely follow actual values, showing the model's capacity to learn complex queuing behavior.

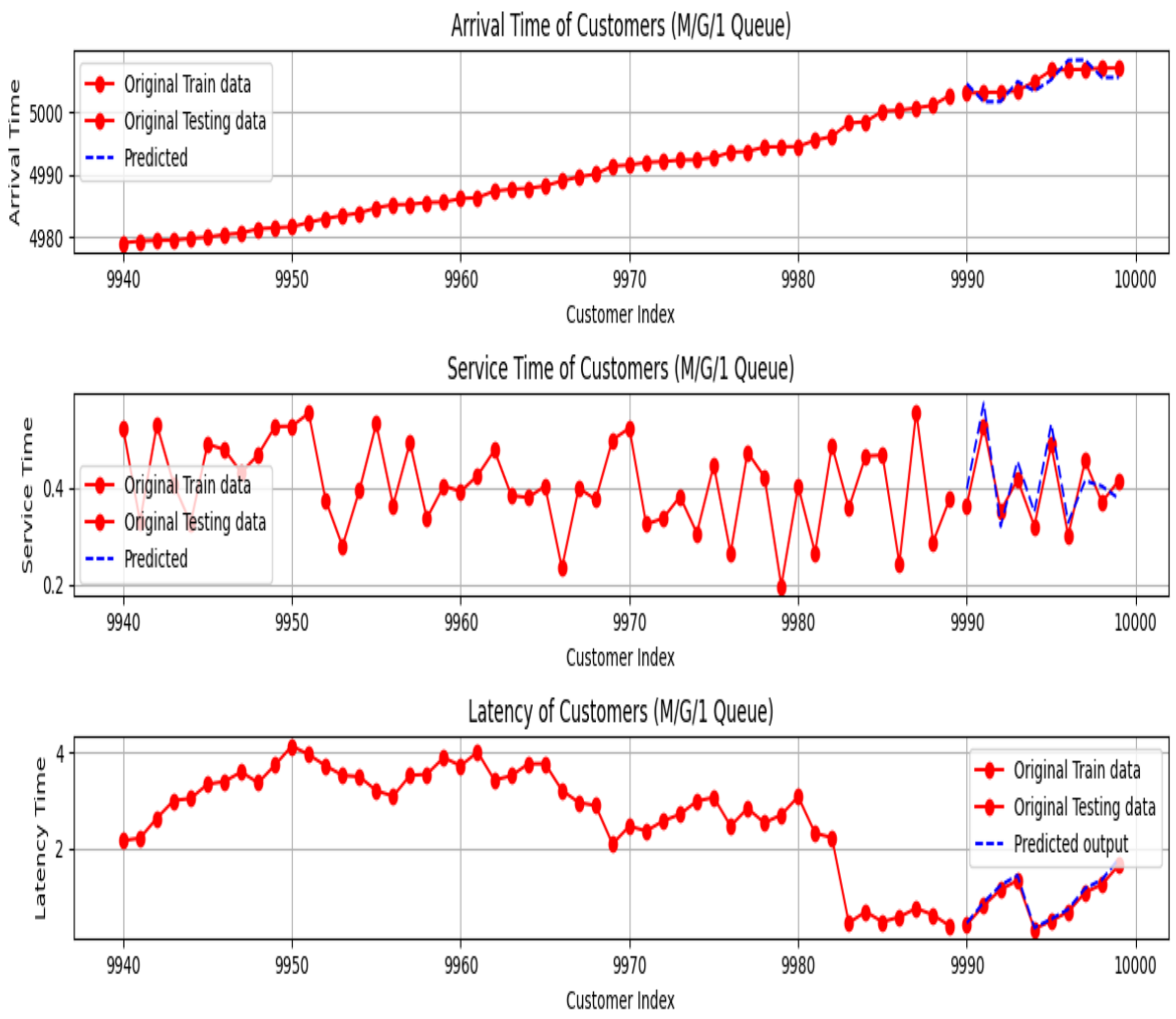


Figure 7.2: Transformer Model Predictions vs Actual for M/G/1 Queue

Figure 7.2 shows how the models capture stochastic variations in service, with Transformer often performing slightly better on sudden service-time bursts.

7.1.3 M/D/1 Queue Model

The M/D/1 queue combines a Poisson arrival process with fixed service times. It blends randomness in arrivals with simplicity in service scheduling.

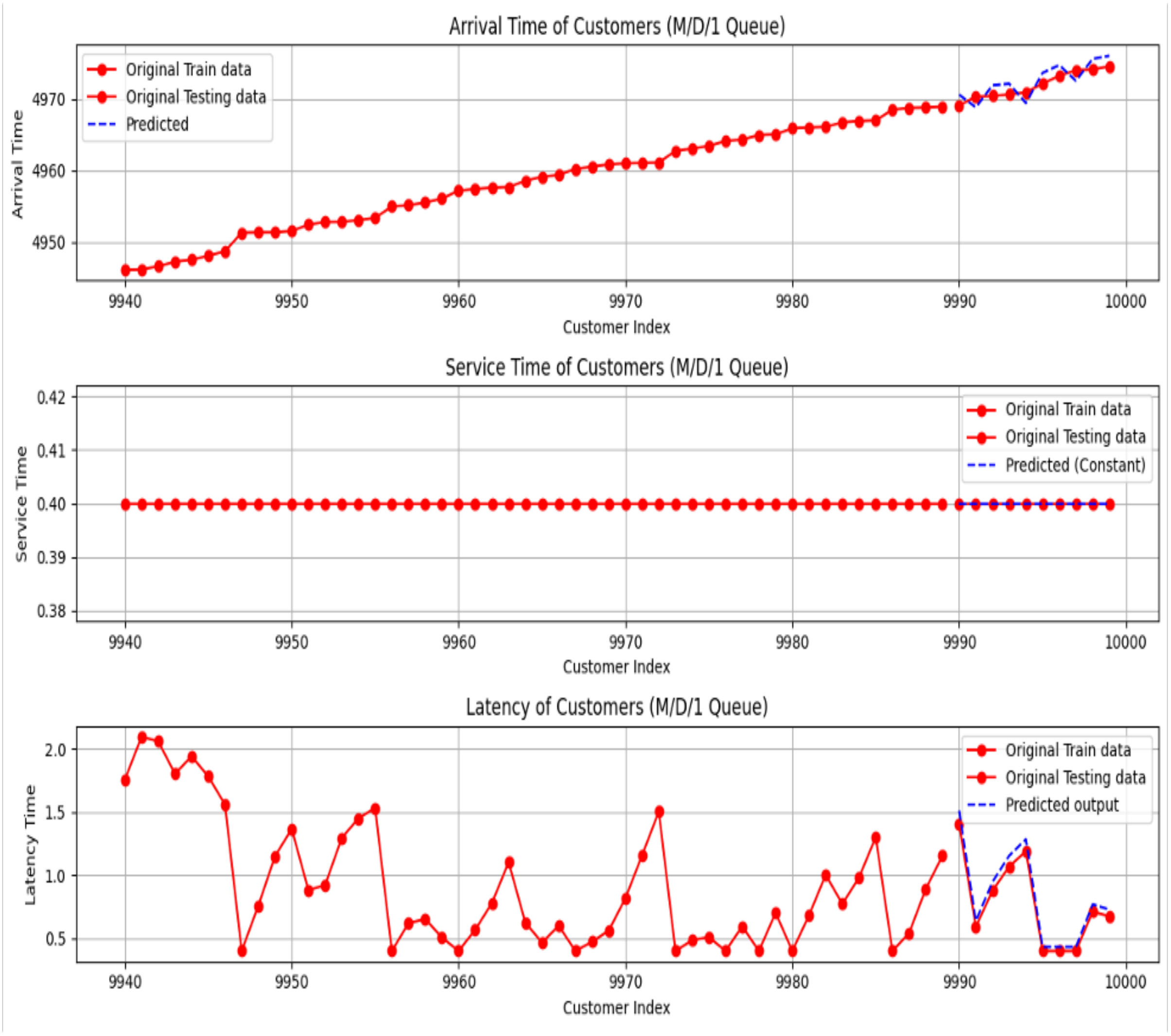


Figure 7.3: Transformer Model Predictions vs Actual for M/D/1 Queue

In Figure 7.3, prediction tracks are mostly smooth, with slight deviation during peak arrival fluctuations. This hybrid nature tests the model's generalization capability.

7.1.4 Aggregator-Based Queueing Model

The Aggregator model represents the summation or combination of multiple traffic sources and queue models. It reflects real-world traffic with high burstiness and non-linear patterns [27].

Key Features:

- Combines multiple arrival/service patterns.
- Captures aggregate traffic behavior from several sources.
- High variability and complex dependencies.

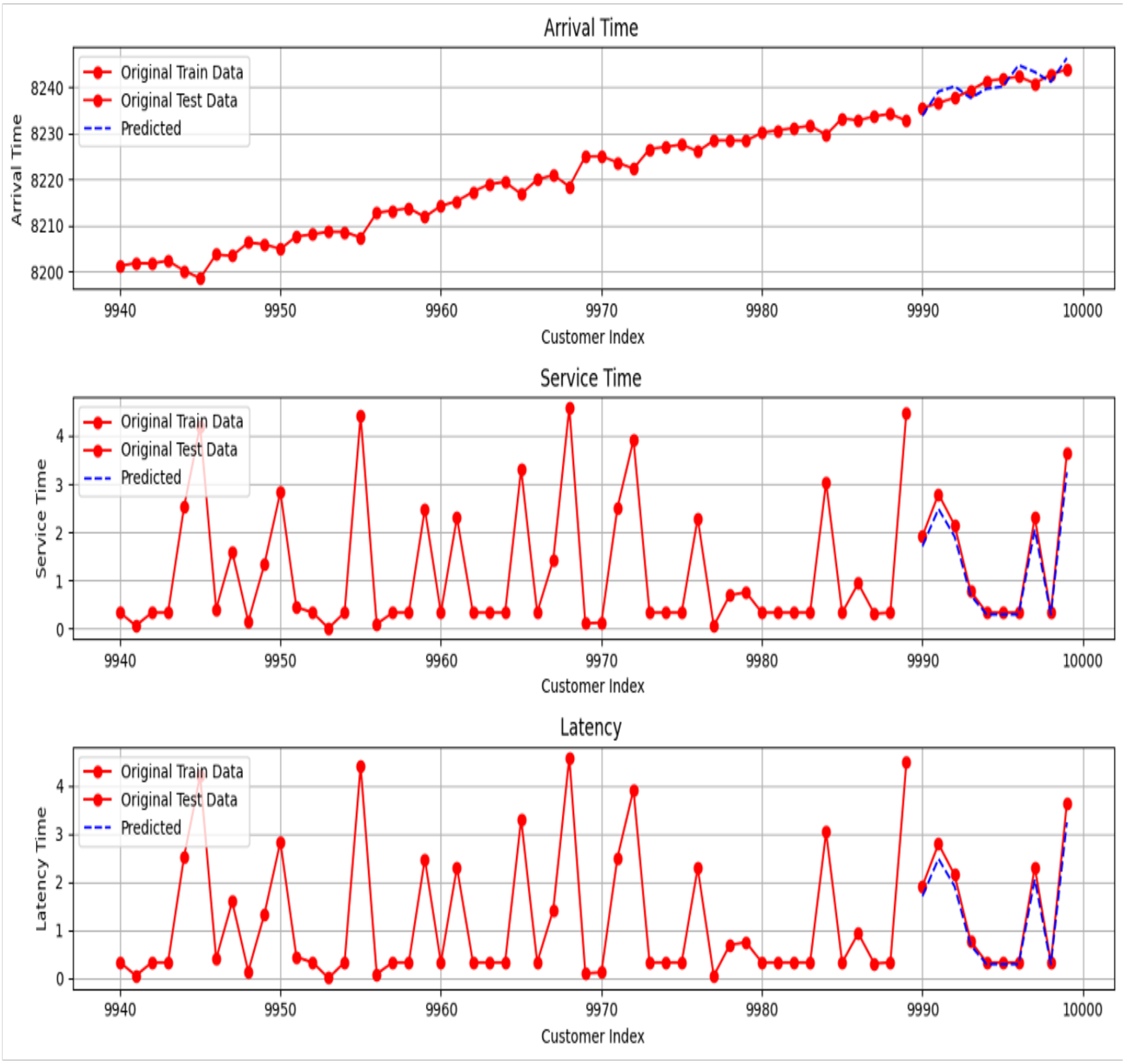


Figure 7.4: Transformer Model Predictions vs Actual for Aggregator Queue Model

As shown in Figure 7.4, this queue type introduces significant prediction challenges. The Transformer model better handles these patterns due to its attention mechanism's capability to model complex dependencies.

7.1.5 M/M/1 Queue Model

The M/M/1 queue is a classical single-server queueing model where:

- Both LSTM and Transformer models capture randomness in arrival and service times.
- The Transformer slightly outperforms LSTM in arrival and service time accuracy.
- Latency prediction is consistent in both models, showing strong learning of queue behavior.

LSTM Model Results

Arrival Time Prediction

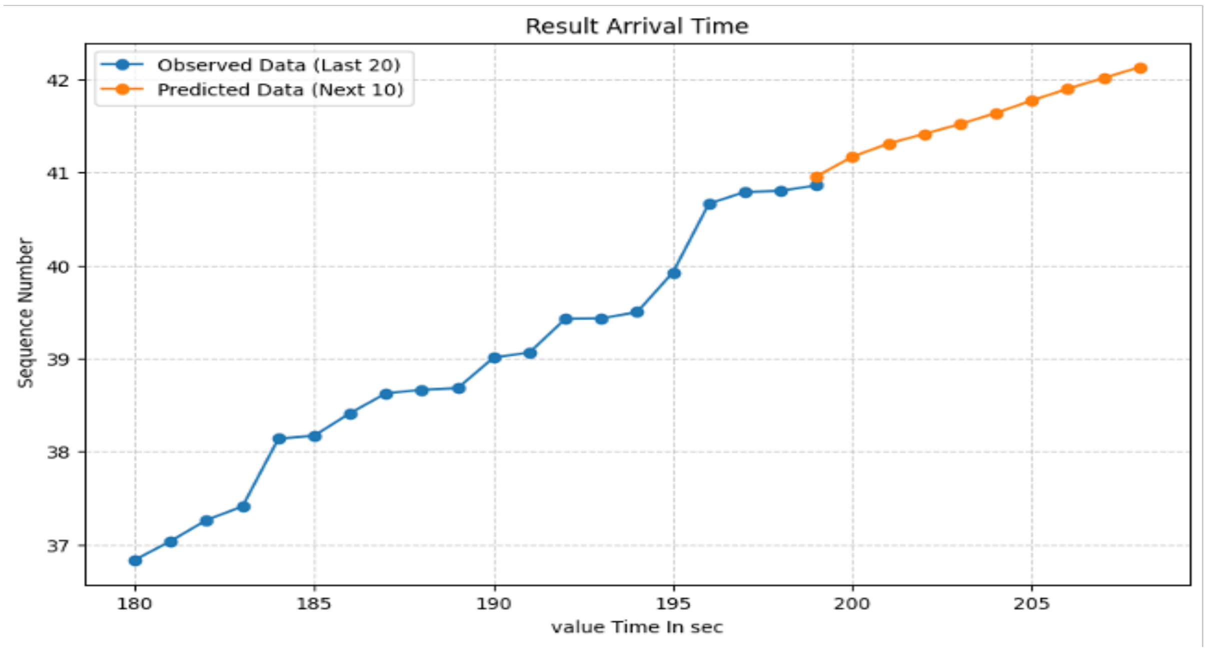


Figure 7.5: LSTM Arrival Time Prediction for M/M/1 Queue

Latency Prediction

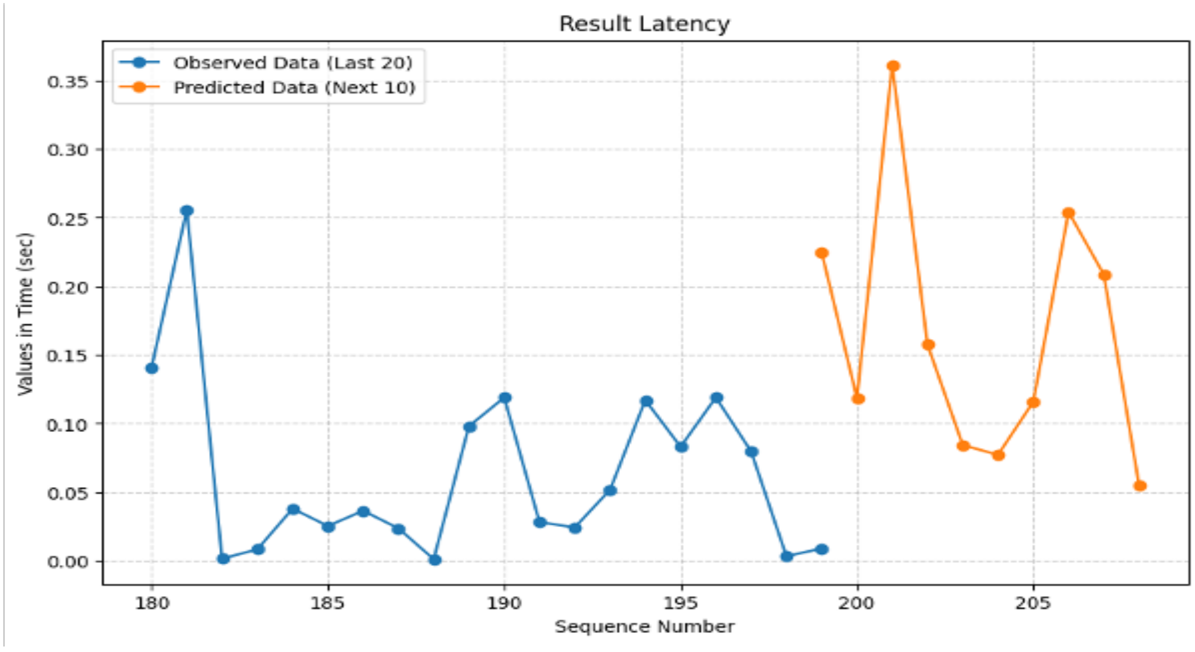


Figure 7.6: LSTM Latency Prediction for M/M/1 Queue

Jitter Prediction

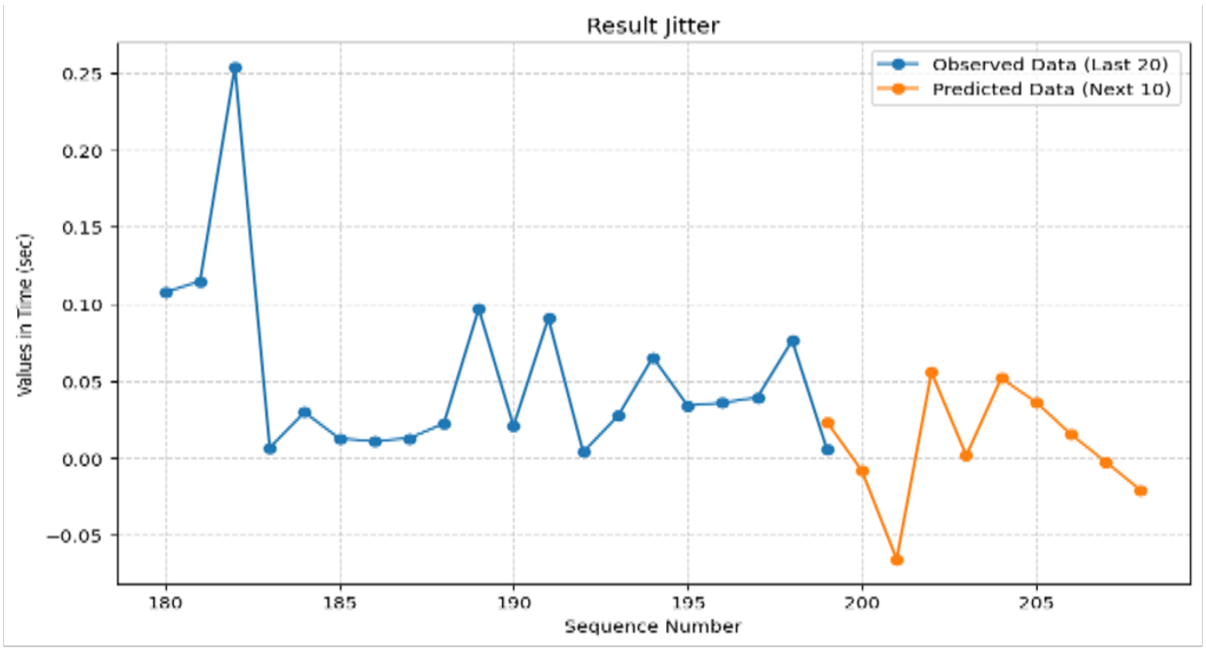


Figure 7.7: LSTM Jitter Prediction for M/M/1 Queue

Transformer Model Results

Arrival Time Prediction

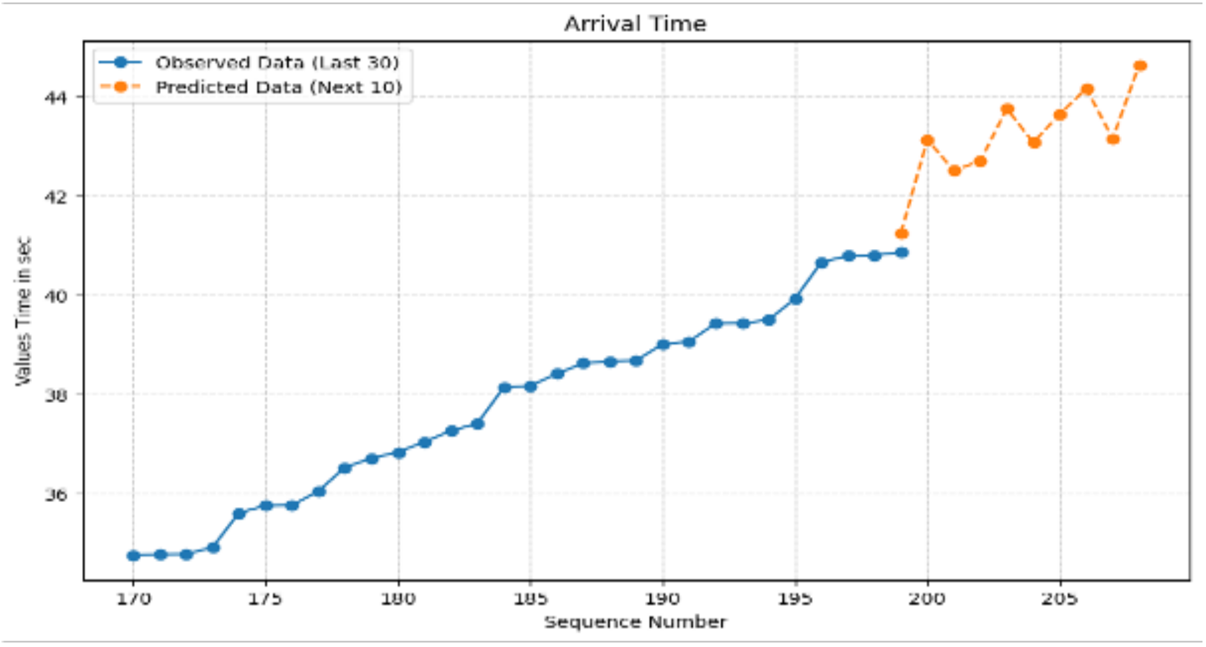


Figure 7.8: Transformer Arrival Time Prediction for M/M/1 Queue

Latency Prediction

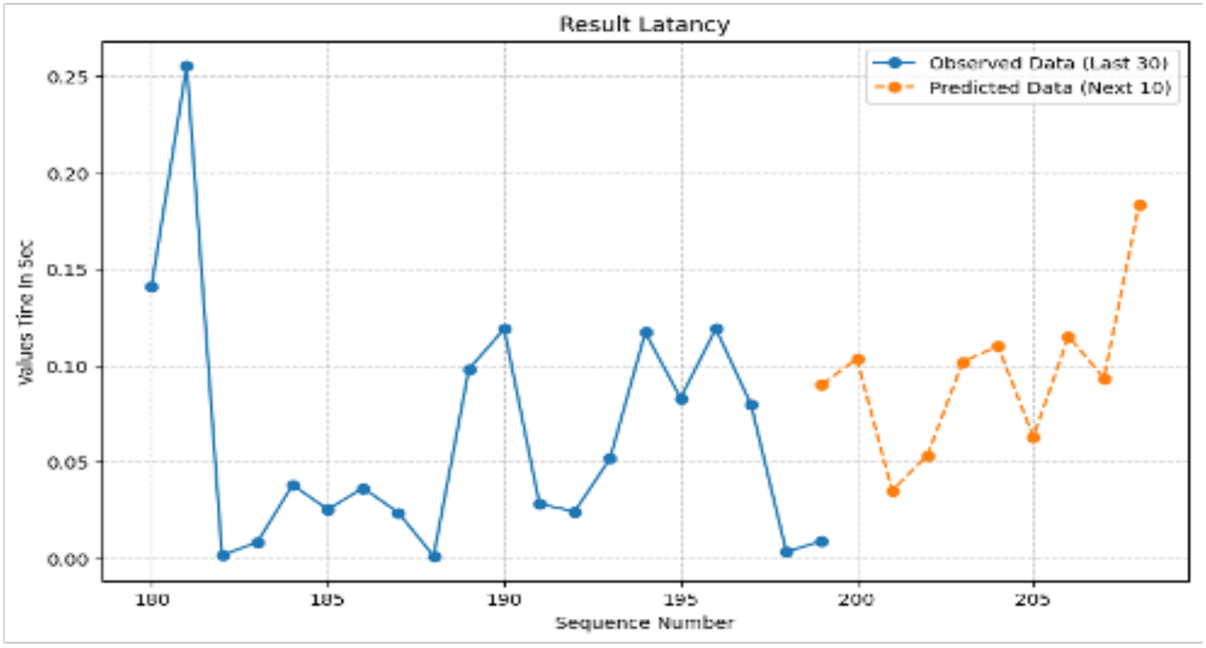


Figure 7.9: Transformer Latency Prediction for M/M/1 Queue

Jitter Prediction

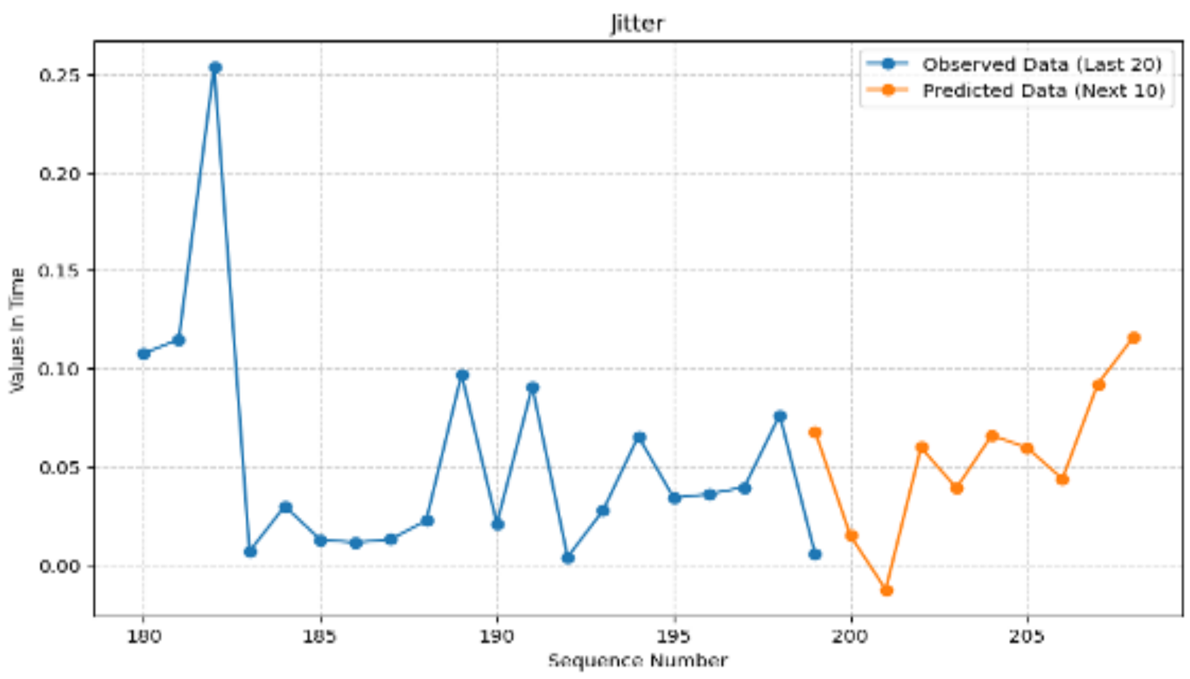


Figure 7.10: Transformer Jitter Prediction for M/M/1 Queue

Analysis

From the plots above, we observe that both LSTM and Transformer models are capable of learning M/M/1 traffic patterns. However, the Transformer generally offers smoother predictions and captures sudden variations (e.g., bursty arrivals or jitter) more accurately. LSTM occasionally lags in response to abrupt changes in queue dynamics.

Training and Validation Loss Analysis

To evaluate the learning behavior of the deep learning models during training, both the training loss and validation loss were recorded across epochs. These loss values provide insights into how well the model generalizes to unseen data. A well-trained model is expected to show a decreasing trend in both losses, with the validation loss eventually stabilizing or slightly increasing due to minor overfitting.

The figure below illustrates the loss curves:

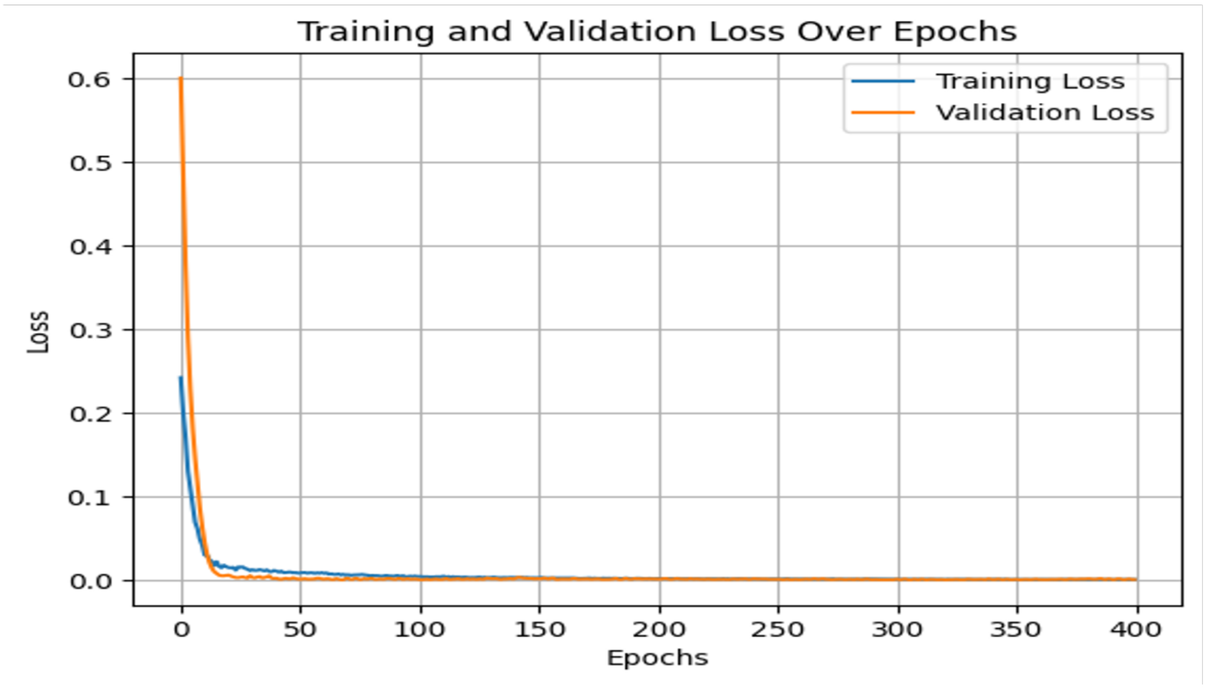


Figure 7.11: Transformer Model Training and Validation Loss over Epochs

Interpretation:

- The training loss consistently decreases, indicating that the model is effectively learning patterns in the data.
- The validation loss follows a similar trend in the initial epochs, confirming that the model generalizes well.
- A small gap between training and validation loss is acceptable and expected, especially in time-series tasks involving queue dynamics.

Summary

The M/M/1 model shows that deep learning can effectively model stochastic queues. Arrival time, latency, and jitter are predicted accurately with both models. The Transformer outperforms slightly in terms of adaptability and consistency.

Table 7.1: **Performance Comparison of Queueing Models Using Various Evaluation Metrics for Transformer Model**

Model	MSE	MAD	MAPE	R^2 Score	RMSE
D/D/1	0.176	0.214	1.87%	0.8889	0.4195
M/G/1	0.153	0.264	2.01%	0.8612	0.3912
M/D/1	0.218	0.374	2.97%	0.7589	0.4289
Router/Aggregator	0.184	0.294	2.69%	0.8589	0.3289

These metrics were computed for four different queueing scenarios: D/D/1, M/G/1, M/D/1, and a real-world Router/Aggregator trace. As shown in Table 7.1, the D/D/1 model achieved the highest R^2 score, indicating highly predictable behavior due to its deterministic nature. In contrast, the M/D/1 model had higher errors, suggesting the difficulty of capturing fixed service patterns amid random arrivals. The Router/Aggregator setup, derived from actual network traces, showed competitive performance, validating the practical applicability of the proposed prediction framework.

Table 7.2: **Performance Metrics for Transformer, LSTM, and Neural Regression Models**

Model	MSE	MAD	MAPE	R^2 Score	RMSE
Transformer	0.45	0.12	0.95%	0.91	0.212
LSTM	1.20	0.25	2.10%	0.76	0.435
Simple Neural Regression	7.50	0.92	8.80%	0.21	0.450

Model-wise Performance Comparison

To identify the most suitable deep learning architecture for queueing behavior prediction, three models were evaluated: Transformer, LSTM, and Simple Neural Regression. Each model was trained on the same dataset, and their performance was measured using standard evaluation metrics including MSE, MAD, MAPE, R^2 score, and RMSE.

The results, as summarized in Table 7.2, show that the Transformer model significantly outperforms both LSTM and Neural Regression. It achieved the lowest MSE (0.45) and highest R^2 score (0.91), indicating high prediction accuracy and strong generalization. LSTM showed moderate performance with acceptable error levels, while the Simple Neural Regression model lagged behind, with the highest error values across all metrics. These findings confirm the Transformer's effectiveness in modeling temporal dependencies and handling variable patterns in queueing systems.

Bibliography

[1] A. Dandoush, V. Priya, M. Uddin, and U. Khalil, "Large Language Models meet Network Slicing Management and Orchestration," *TechRxiv Preprint*, 2024. doi: 10.36227/techrxiv.171173194.46729464/v1.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting," *ICLR*, 2018.

[4] J. Feng et al., "Deep Learning Based Traffic Prediction and Its Application on Resource Allocation in Wireless Networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2807–2817, 2019.

[5] Y. Zhang and Y. Zheng, "Traffic Prediction with Transformer and Graph Attention Networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5146–5155, 2020.

[6] Y. Cheng, Q. Sun, and B. Zhang, "Network Traffic Prediction Based on LSTM and Attention Mechanism," *IEEE Access*, vol. 9, pp. 107180–107192, 2021.

[7] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, 2017.

[8] Y. Huang, Y. Zhang, Y. Zhang, Z. Guo, H. Wang, J. Zhang, and K. Chen, *Large Language Models for Networking: Applications, Enabling Techniques, and Challenges.* arXiv preprint arXiv:2311.17474, 2023.

[9] M. Dandoush, R. Ferrus, O. Sallent, J. Pérez-Romero, and P. Legg, *Large Language Models Meet Network Slicing Management and Orchestration.* arXiv preprint arXiv:2403.13721, 2024.

[10] Kammoun, I., Ajmi, I., Tabbane, N. (2019). Proactive Network Slices Management Using Fuzzy Logic and SVR. In 2019 International Conference on Wireless Networks and Mobile Communications (WINCOM).

[11] Huang, K., He, Z., Li, X., Liu, L., Guo, Y., Zhang, H., ... & Liu, J. (2023). Large Language Models for Networking: Applications, Enabling Techniques, and Challenges. arXiv preprint arXiv:2311.17474.

[12] Dandoush, H., Gomes, F., Zardini, A., De Cola, T., Kassler, A., & Bernal, M. (2024). Large Language Models Meet Network Slicing Management and Orchestration. arXiv preprint arXiv:2403.13721.

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[14] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT.

[15] Bariah, L., Rajatheva, N., Ylianttila, M., & Latva-aho, M. (2023). Large Language Models for Telecom: The Next Big Thing?. IEEE Communications Magazine.

[16] Chen, J., Liu, Y., Zhang, Y., & Xu, K. (2023). NetGPT: A Native-AI Network Architecture. arXiv preprint arXiv:2306.09130.

[17] Miao, Y., Ding, N., Zhu, C., & Fu, J. (2023). An Empirical Study of NetOps Capability of Pre-trained LLMs. arXiv preprint arXiv:2305.17680.]

[18] Kammoun, I., Ajmi, I., Tabbane, N. (2019). Proactive Network Slices Management Using Fuzzy Logic and SVR. In 2019 International Conference on Wireless Networks and Mobile Communications (WINCOM).

[19] S. Li et al., "Enhancing the locality and breaking the memory bottleneck of Transformer on time series forecasting," Advances in Neural Information Processing Systems, vol. 32, 2019.

[20] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.

[21] H. Wu et al., "Deep Transformer models for time series forecasting: The influenza prevalence case," arXiv preprint arXiv:2001.08317, 2020.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] C. M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[24] L. Kleinrock, Queueing Systems, Volume 1: Theory, Wiley-Interscience, 1975.

[25] E. Hossain and M. S. Hasan, "Toward 5G: A survey on 5G network architecture and emerging technologies," IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1658–1686, 2019.

[26] Q. Li et al., "Intelligent 5G: When cellular networks meet artificial intelligence," IEEE Wireless Communications, vol. 27, no. 5, pp. 76–83, 2020.

[27] Z. Yang et al., "A survey of deep learning techniques for mobile traffic forecasting," IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1674–1693, 2017.

[28] S. Garcia, J. Luengo, and F. Herrera, Data Preprocessing in Data Mining, Springer, 2016.

[29] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, Elsevier, 2011.

[30] Q. Wen et al., "Transformers in Time Series: A Survey," arXiv preprint arXiv:2202.07125*, 2022.

[31] J. Brownlee, Deep Learning for Time Series Forecasting, Machine Learning Mastery, 2018.

[32] C. Bergmeir and J. M. Benitez, "On the use of cross-validation for time series predictor evaluation," Information Sciences, vol. 191, pp. 192–213, 2012.

[33] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, vol. 13, pp. 281–305, 2012.

[34] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action, Cambridge University Press, 2013.

[35] G. Bolch et al., Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Wiley, 2006.

[36] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," Climate Research, vol. 30, no. 1, pp. 79–82, 2005.

[37] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," Advances in Neural Information Processing Systems, vol. 32, 2019.

[38] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," OSDI, vol. 16, pp. 265–283, 2016.

[39] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," Nature Methods, vol. 17, pp. 261–272, 2020.