Received: 20 July 2025 | Revised: 9 January 2026 | Accepted: 2 February 2026 | Published online: 15 February 2026

RESEARCH ARTICLE

Artificial Intelligence and Applications
2026, Vol. 00(00) 1–15
DOI: 10.47852/bonviewAIA62026867

BON VIEW PUBLISHING

An Enhanced Traffic Matrix Prediction in Software-Defined Networks Using Bi-LSTM with Attention Mechanism

Prabu. U.[1,*], Prabhala Harichandana[1], Manepalli Baby Kavyasri[1], Uppala Bhargavi[1] and Geetha. V.[2]

[1]*Department of Computer Science and Engineering, Siddhartha Academy of Higher Education (Deemed to be University), India*

[2]*Department of Information Technology, Puducherry Technological University, India*

Abstract: Software-defined networks (SDNs) play a vital role in network traffic management, traffic estimation, traffic engineering, and routing. The effective management of SDNs relies on traffic matrix prediction (TMP). The traffic matrix (TM) provides a comprehensive view of each origin–destination flow. An SDN paradigm offers various techniques to obtain additional information about the TM. This information is useful for managing network operations, such as performance diagnostics, traffic engineering, and network design. The TMs are critical for network operation and management. However, TMP remains a challenging task due to the dynamic and complex nature of network traffic patterns. To address this challenge, a Bi-LSTM model with attention mechanism is proposed for better prediction. The proposed model captures both forward and backward temporal dependencies in historical traffic data that enable the model to learn short-term and long-term traffic correlations, while the attention mechanism focuses on critical timesteps that help the model learn the traffic behavior and traffic patterns more effectively. The proposed model is experimented with real-world topologies such as Abilene, GEANT, Nobel-Germany, and Germany50. The TMs of these topologies are in the flattened form with different granularities and horizons. The performance of the proposed model is evaluated using metrics such as mean squared error, mean absolute error, coefficient of determination, and average inference time. The experimental results show that the Bi-LSTM model with attention mechanism has outperformed the Bi-LSTM with Adam model in terms of accuracy with better TMP in all considered real-world topologies.

Keywords: software-defined networks, traffic matrix prediction, Bi-LSTM, attention mechanism

1. Introduction

Software-defined networks (SDNs) are a novel approach to network architecture that separates the control plane from the data plane to provide centralized network management. Typically, in traditional networks, routing decisions entirely depend on individual routers. Routers and switches cannot be programmed because they rely on static routing tables. SDNs overcome these limitations by introducing a centralized controller that makes centralized decisions. In SDNs, it is important to understand the traffic behavior between nodes for network management. This understanding can be achieved through the prediction of traffic matrices (TMs). Traffic Matrix V represents the traffic volume between different source and destination nodes [1]. Traffic matrix prediction (TMP) involves predicting the traffic volume between origin–destination (OD) pairs using historical TMs. This process is typically carried out using statistical methods or machine learning techniques that are capable of capturing the temporal and spatial dependencies in network traffic. A TMP model aims to predict future network traffic values using past TMs. TMP is crucial for resolving network issues, such as routing and resource allocation [2]. TMP is more accurate and efficient in SDNs due to its centralized control and programmability. This enables the measurement of a wider range of flows, opening up new possibilities for solving the challenges of TMP [2]. The data plane of SDNs provides ternary content addressable memory (TCAM) entries in the flow table for TMP, where the traffic values are collected from switches in SDNs [1]. The network administrator requires insights into the TM, which plays a key role in the design, control, and management of network services such as traffic engineering (TE), traffic monitoring, and network management [3]. TMP helps network administrators detect issues such as congestion, packet loss, and bottlenecks.

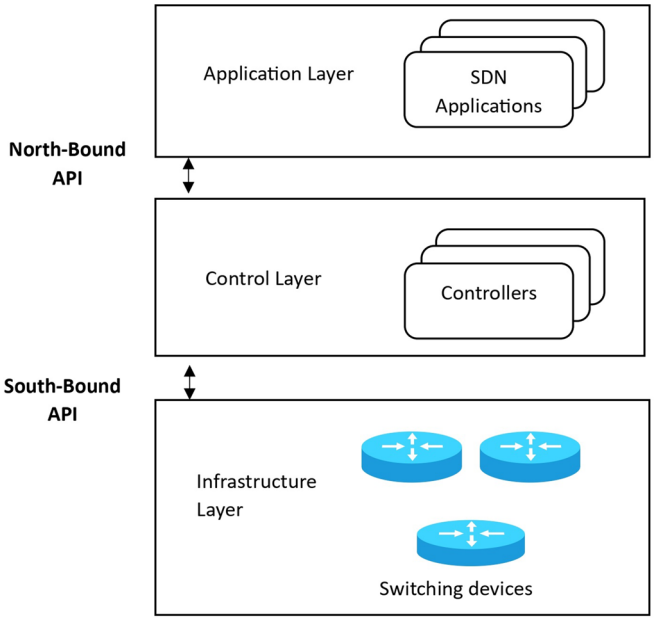
1.1. Software-defined networks

SDNs introduce programmability for static switches, which makes it easy for the controller to manage, optimize, and configure the network. The SDN architecture primarily consists of three layers as shown in Figure 1: the application layer, where all client programs reside; the control layer, also called as the control plane, which consists of a centralized controller; and the infrastructure layer, also called as the data plane, which consists of dumb switches and routers. The client applications in the application layer request network services to the control plane through the northbound API and the controller converts the network protocol generated in the control plane to rules that are programmed to switches through routers using the southbound API. By collecting traffic data from the SDN switches, the controller in the control plane can get a complete view of the network's resources and monitor each flow [1]. Thus, it contributes to accurate TMP by eliminating the need to equip the network

*Corresponding author: Prabu. U., Department of Computer Science and Engineering, Siddhartha Academy of Higher Education (Deemed to be University), India. Email: prabu@vrsiddhartha.ac.in

instruments, allowing the addition of new flow statistics to link counts [1]. However, the number of entries in a flow table for traffic measurement is limited because SDN nodes typically use expensive TCAM to build their flow tables [2]. The SDN controller configures the router and switches, which makes traffic value measurement easy [2]. However, configuring switches through a controller, especially TCAM, becomes expensive while handling a large number of flows, and it is limited in size [3]. Due to this limitation, it is challenging to achieve balance between traffic measurement and the scalability of the system.

Figure 1
SDN architecture



1.2. Traffic matrix prediction

A TM represents the traffic volume measured between different nodes. It gives a detailed presentation of traffic flow from origin to destination nodes [1]. Predicting each TM in the network is essential for the routing strategy setup, infrastructure allocation, traffic distribution, and irregularity detection [4]. However, the cost of predicting TMs is high as the network size increases [5]. By considering a network with k nodes and l links, the number of possible flows between each node is k^2. Assume that at any time step t, the typical representation of the TMs is given in Equation (1) [5].

$$V_t = [V_1(t), V_2(t), \ldots\ldots, V_N(t)], \tag{1}$$

where V_t represents the TM that is flattened into a vector at time step t and each element $V_i(t)$ is the traffic volume between a specific OD node pair. This vector captures all possible flows between nodes.

TMP involves taking past TMs to predict the future TMs. The number of links between nodes might be less than the total possible flows, which makes the TMP a linear inverse problem [5]. The core problem of TMP is solving an inverse problem that is built by link counts, routing information, and the TMs [6]. It is a challenging task to predict accurate TMs for any network topology. Although SDN provides flexibility and centralized control, many TMP methods rely on traditional load links.

1.3. Long short-term memory

Long short-term memory (LSTM) is a recurrent neural network (RNN) that keeps a record of the long-term and short-term memories of past data [7]. Many network models cannot retain the information from the previous timestep, but LSTM is specifically designed to address this limitation. LSTM extends the RNN's memory to capture the long-term dependencies in the input data [7]. LSTM remembers the historical data through the gating mechanism. The architecture consists of three types of gates, namely, the input gate, forget gate, and output gate. These gates can remember or forget the information by updating the cell states [8]. Each storage unit in the architecture consists of the above mentioned three gates [8]. As the gates are incorporated in LSTM to remember past and current data, the input is traversed only in one direction, i.e., from left to right [7]. It remembers only what is worth being preserved from the input data; if not needed, it removes those data from the memory called "cell state" [7]. Among the three gates of LSTM, the forget gate decides to remove or remember the data, the input gate decides how much of the input data must be given to cell state so that the model focuses on critical information, and the output gate decides whether the existing value in a cell affects the output. The architecture of LSTM is illustrated in Figure 2, where the input data and previous hidden state output x_{t-1} are given as inputs along with the previous cell state C_{t-1}. The sigmoid function is applied to x_t along with h_{t-1}, and this output passes through the forget gate to determine which information is important. These data are combined with the previous cell state. x_t and h_{t-1} are passed through the sigmoid function to form the input gate and through the tanh function to form candidate values. These are combined to update the cell state, allowing the LSTM to retain important temporal information. The internal working principle of the three gates is described by the below formulas.

The computation at the forget gate to decide which information to retain from the previous cell state at the current timestep is shown in Equation (2).

$$f_t = \sigma\left(w_f \cdot \left[h_{t-1}; x^t\right] + b_f\right). \tag{2}$$

At the input gate, input data i_t and candidate cell state $C_t^{\sim}$ are computed as shown in Equations (3) and (4), respectively

$$i_t = \sigma(w_{i_h}[h_{t-1}], w_{i_x}[x_t], b_i), \tag{3}$$

$$C_t^{\sim} = tanh(w_{C_h}[h_{t-1}], w_{C_x}[x_t], b_c). \tag{4}$$

At the output gate, output gate activation $\boldsymbol{O_t}$ and hidden state output $\boldsymbol{h_t}$ are calculated as shown in Equations (5) and (6), respectively

$$O_t = \sigma(w_{O_h}[h_{t-1}], w_{O_x}[x_t], b_O), \tag{5}$$

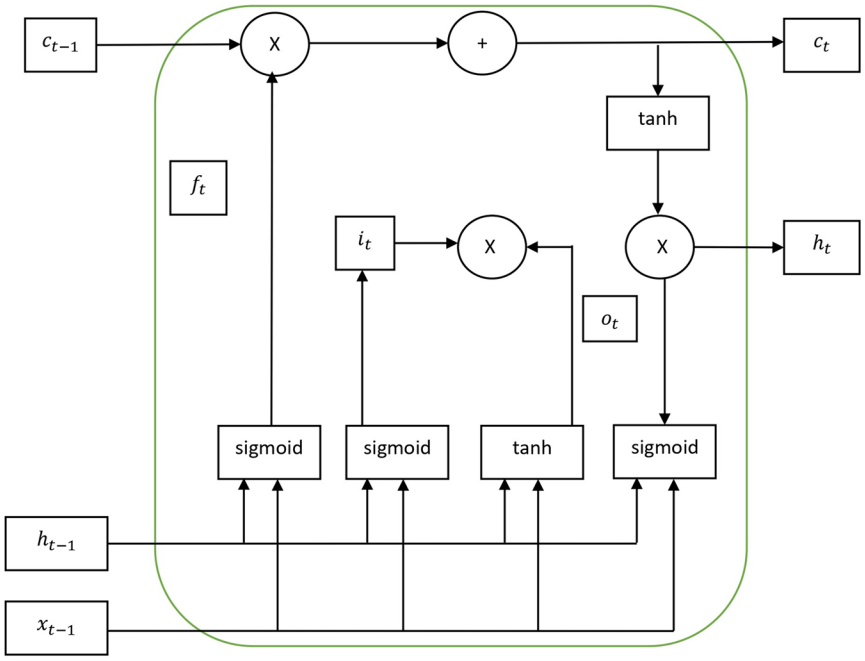
$$h_t = O_t.tanh(c_T). \tag{6}$$

where

f_t is the output of the forget gate,
w_f and B_f are the weight and bias matrices, respectively,
x_t is the input value at time "t,"
h_{t-1} is the output of the hidden layer at time "$t-1$,"
i_t represents the data needed to be given as input or not and,
$C_t^{\sim}$ represents the vector of newly added values,
O_t is the output from the output gate.

LSTM connects the memory through the cell state and hidden state for maintaining continuity in the network [9].

Figure 2
Architecture of LSTM



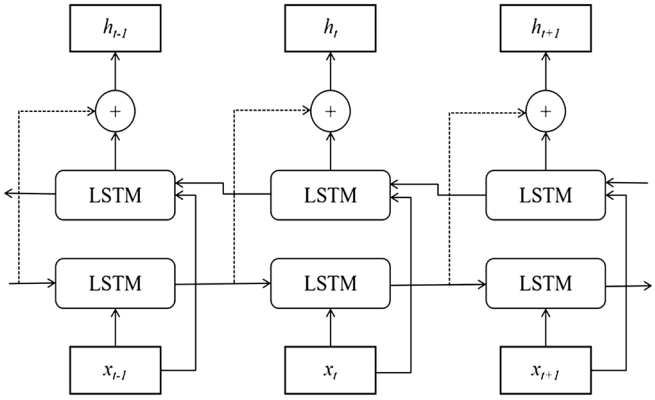
1.4. Bi-directional LSTM

Bi-directional LSTM (Bi-LSTM), an extended version of LSTM, introduces both the forward pass and the backward pass. It combines both forward LSTM and backward LSTM [10]. Bi-LSTM makes better predictions than LSTM because the LSTM model is applied to input data in the forward pass and to reverse the input data in the backward pass [10]. This feature of Bi-LSTM allows learning from past and future data [11]. This also increases the overall accuracy of the model because it uses the input data twice in the training [10]. Therefore, the Bi-LSTM can extract the bi-directional temporal features from the input data [11]. Features such as dual layers, dropout, and multi-head make this model proficient in capturing the complex data dependencies [11]. Figure 3 depicts the architecture of Bi-LSTM [10]. The forward pass, backward pass, and their combination in Bi-LSTM are described in Equations (7), (8), and (9):

Forward pass:

$$\overrightarrow{h_t}, \overrightarrow{C_t} = LSTM\left(x_t, \overrightarrow{h_{t-1}}, \overrightarrow{c_{t-1}}\right). \tag{7}$$

Figure 3
Bi-LSTM architecture



Backward pass:

$$\overleftarrow{h_t}, \overleftarrow{C_t} = LSTM\left(x_t, \overleftarrow{h_{t-1}}, \overleftarrow{c_{t-1}}\right). \tag{8}$$

Concatenation of forward and backward hidden states:

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t}. \tag{9}$$

1.5. Attention mechanism

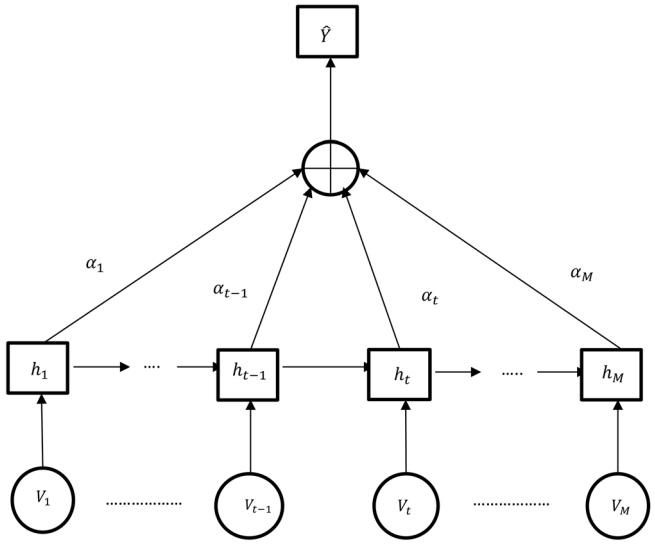
An attention mechanism is used for resource allocation and for optimization of weights and biases in the context of TMP. It acts as a human brain with attention, which leads to focused learning [10]. It focuses on important data and discards useless data [12]. In large-scale traffic data, with many flows from source to destination with fluctuating values, it becomes essential to identify which spatial or temporal inputs significantly contribute to the TMP. It focuses on critical information that increases the model's overall accuracy [10]. The core implementation of the attention mechanism lies in shifting the attention toward important information, ignoring the irrelevant information, and extending the necessary information to a large extent [12]. Figure 4 shows that the input vector V_t, is passed through a Bi-LSTM unit, producing hidden states h_1, h_{t-1}, h_t, and h_M, which are then passed to the attention mechanism, where attention weights α_1, α_{t-1}, α_t, and α_M are assigned to each hidden state. It represents the importance of the hidden state for producing the final output. $\oplus$ represents the context vector c, which is the weighted sum of all hidden states. Y is the final output of the attention mechanism.

The formula for attention weights is shown in Equation (10):

$$\alpha_i = \frac{e^{e_i}}{\sum_{j=1}^{n} e^{e_j}}, \tag{10}$$

where e_i is the score measuring relevance between the current decoding context and the hidden state h_i.

Figure 4
Attention mechanism



A context vector is shown in Equation (11):

$$c = \sum_{i=1}^{n} \alpha_i * h_i. \tag{11}$$

Final prediction is shown in Equation (12):

$$\widehat{Y} = w_{f_c} * c_{drop} + b, \tag{12}$$

where

C_{drop} is the context vector after passing c through the dropout layer,
w_{f_c} is the assumed weight vector.

Example: Consider that there are three hidden layers h_1, h_2, and h_3. Now, we need to compute the context vector and the final prediction $\hat{Y}$.

Assume that $h_1 = \begin{bmatrix}0.2\\0.4\end{bmatrix}$, $h_2 = \begin{bmatrix}0.6\\0.1\end{bmatrix}$, $h_3 = \begin{bmatrix}0.5\\0.3\end{bmatrix}$.

Let the query vector be $q = \begin{bmatrix}0.4\\0.6\end{bmatrix}$.

Step 1: Calculation of attention score $e_i = q^T * h_i$

$$e_1 = [0.4 \quad 0.6] * [0.2 \quad 0.4] = 0.08 + 0.24 = 0.32$$

$$e_2 = [0.4 \quad 0.6] * [0.6 \quad 0.1] = 0.24 + 0.06 = 0.30$$

$$e_3 = [0.4 \quad 0.6] * [0.5 \quad 0.3] = 0.20 + 0.18 = 0.38.$$

Step 2: Calculation of attention weights: $\alpha_i = \frac{e^{e_i}}{\sum_{j=1}^{n} e^{e_j}}$

$$e^{0.32} \approx 1.377, e^{0.30} \approx 1.350, e^{0.38} \approx 1.462$$

Sum: Z = 1.377 + 1.350 + 1.462 = 4.189

$$\alpha_1 = \frac{1.377}{4.189} \approx 0.329$$

$$\alpha_2 = \frac{1.350}{4.189} \approx 0.322$$

$$\alpha_3 = \frac{1.462}{4.189} \approx 0.349.$$

Step 3: Calculate the context vector: $c = \sum_{i=1}^{n} \alpha_i * h_i$

$$c = 0.329 * h_1 + 0.322 * h_2 + 0.349 * h_3$$
$$= 0.329 * \begin{bmatrix}0.2\\0.4\end{bmatrix} + 0.322 * \begin{bmatrix}0.6\\0.1\end{bmatrix} + 0.349 * \begin{bmatrix}0.5\\0.3\end{bmatrix}$$
$$= \begin{bmatrix}0.434\\0.269\end{bmatrix}.$$

Step 4: Calculate the final prediction:

$$\widehat{Y} = w_{f_c} * c_{drop} + b.$$

Assume that b = 0.1 and $w_{f_c} = [1 \quad 2]$

$$\widehat{Y} = [1 \quad 2] * \begin{bmatrix}0.439\\0.269\end{bmatrix} + 0.1 = 0.4344 + 0.538 + 0.1 = 1.072,$$

where

V_t represents the input data,
h_t is the hidden layer output,
α_t is the attention probability distribution value given as input by attention to the hidden layer, after optimizing the weights and biases,
y is the output of the network, computed by the attention mechanism.

The major contributions of this article are as follows:

1) The traffic demand in TMs is normalized using MinMax normalization.
2) The TM is predicted using Bi-LSTM with attention mechanism.
3) The proposed model is evaluated using four different topologies.
4) The proposed work is compared with existing Bi-LSTM with Adam.

2. Literature Review

Zhao and Tan [13] proposed the principal component analysis optimization method (PCAOM), a TM estimation approach for IP networks. Because direct measurement of traffic is challenging, their approach uses existing link data and known routing paths to infer traffic flows. PCAOM models the TM estimation problem as an optimization task that minimizes the Mahalanobis distance between the estimated and expected TMs, which helps in accounting for data correlations and improving accuracy. To reduce dimensionality and computation, the model applies principal component analysis (PCA) to reduce the complexity of OD traffic data while preserving principal flow patterns. The estimation is solved using the Moore–Penrose inverse, eliminating the need for iterative algorithms. PCAOM introduces a tunable weight parameter $\gamma \in [0, 1]$ to balance the influence of prior information depending on its reliability. The method was tested on the Abilene dataset and compared to four baseline models: Tomo-Gravity, SRMF, standard PCA, and neural network methods. PCAOM reduced spatial relative error (SRE) by up to 27.7% and temporal relative error (TRE) by up to 22.9% using the Mahalanobis distance model when $\gamma = 0.2$. It showed better accuracy and handled noisy data well, although it needed careful adjustment of the γ value for best results.

Nie et al. [14] worked on predicting and estimating TMs in data center networks using deep learning techniques. Two separate methods are proposed using deep belief networks (DBNs). For prediction, they designed a setup with a DBN paired with a logistic regression model to predict future traffic based on past patterns. It finds how traffic changes over time for each OD pair, learning from a series of past data points to predict the next one. For estimation, they used another DBN to work out the TM from link counts, handling the tough problem where there are more OD pairs than the links to measure. These two proposed methods

are tested on two real-time traffic datasets (one jumpy and one steady) and compared to three methods: SRMF, Tomo-Gravity, and PCA. For prediction, their DBN method outperformed the models SRMF, Tomo-Gravity, and PCA, reducing errors by 53.1%, 47.9%, and 83.6% in the jumpy dataset and by 9.8%, 5.7%, and 52.4% in the steady one. For estimation, it improved errors by 32.7%, 25.3%, and 23.4% in the jumpy set and 47.7%, 45.7%, and 72.0% in the steady set. It showed low bias for both tasks, especially in the jumpy data, although its variance was a bit higher than SRMF's in prediction.

Nie et al. [15] proposed a deep learning method based on a DBN to predict and estimate TMs in large-scale IP backbone networks. The method uses a DBN to capture temporal and statistical patterns in traffic data for prediction and to solve the inverse inference problem in TM estimation. The techniques include data normalization for scaling of traffic data to a [0,1] range to suit the binary nature of DBNs and a sliding window for prediction that constructs training datasets from historical traffic data. The DBN is composed of stacked restricted Boltzmann machines (RBMs), trained in a greedy layer-wise manner to learn traffic features. For TM estimation, the DBN takes link counts and routing information as inputs. The output is then corrected using an iterative proportional fitting (IPF) algorithm to satisfy network constraints. The method was tested on the Abilene and GEANT datasets. The performance is measured using bias and standard deviation. Compared to the PCA method, the DBN reduced prediction bias by approximately 25% and standard deviation by 20% for Abilene, and estimation bias by approximately 30% and standard deviation by 25% for Abilene, with similar improvements for GEANT, particularly for short-term traffic variations.

Emami et al. [16] proposed a new approach to estimate end-to-end traffic using graph embedding and convolutional neural networks (CNN). This approach treats the network like a graph that changes over time. The approach builds load adjacency matrices (L2AMs) using information about link utilization levels and how the network is laid out. These matrices are then used by CNN, which is good at finding patterns and relationships. The CNN is used to understand how different link loads affect each other. Using L2AMs, the CNN gets a clear view of how the network is structured and how traffic flows through it. This helps it estimate traffic between different start and end points (OD pairs) without needing to guess the actual paths that the data take. The proposed approach CNTME was tested on the Abilene dataset, against two neural network models: MNETME and BPTME. CNTME had a median SRE of 0.4972, 16% lower than MNETME's 0.5917 and 21.5% lower than BPTME's 0.6327. Its median temporal relative error (TRE) was 0.3746, 25% better than MNETME's 0.4996 and 45.5% better than BPTME's 0.6879. These results are evaluated from 1000 time slots. CNTME also followed traffic swings better for random OD flows.

Kumar et al. [17] proposed a multi-view subspace learning approach for accurate TM estimation using canonical correlation analysis. The proposal involves building multiple "views" of traffic data—basically different takes on the same network traffic using simple, low-cost methods. Four views labeled A, B, C, and D from techniques called gravity, generalized gravity, tomography, and generalized tomography are created. These views get paired up into two sets, namely, {A, B} and {C, D}, matching them by time, and CCA figures out a shared space where these pairs line up best. The method is tested against four other single-view approaches: PCA, Fanout, {1}-inverse, and RTME, and using 19 weeks of real traffic data from the Abilene network (X 06–X 24). The results showed that their CCA-based approach was a standout: it brought the relative temporal error (RTE) down to under 0.16 (better than the 0.27 to 0.62 of the others) and the relative spatial error (RSE) to below 15 0.07 (compared to 0.30 to 0.70), cutting errors by over 80%. For one week's data, X19, it scored an RTE of 0.05 and RSE of 0.06, while PCA struggled with 0.49 and 0.45. In addition, it held steady with almost no bias or variance and finished in under 0.26 seconds.

Memon et al. [18] proposed R-CNTME, a CNN-based TM estimator for cloud networks. The problem addressed is estimating TMs accurately to support network management tasks such as anomaly detection and capacity planning, especially when training data are limited, sparse, or noisy. R-CNTME improves TM estimation by modifying the existing CNTME architecture to better handle real-world imperfections. The approach has two key components: graph embedding, which transforms one-dimensional link load data into a two-dimensional link adjacency matrix (L2AM), and a rewired CNN architecture, which flattens feature matrices after convolution layers to create a single system-wide feature vector. Unlike traditional CNNs using MSE, R-CNTME uses the Softplus activation function to avoid negative flow estimates. It was evaluated on the Abilene dataset using an Adam optimizer, with performance measured using SRE, TRE, bias, and standard deviation. Compared to the baseline CNTME, R-CNTME showed a 30.7% better TRE at sparsity 0.2 and 13.6% better SRE on clean data, especially when 30% of L2AM entries had simulated noise.

Aloraifan et al. [19] proposed two hybrid deep learning models for network TMP to improve network management and planning. The first model integrates CNN, LSTM, and Bi-LSTM, while the second combines CNN, gated recurrent units (GRU), and bidirectional GRU (Bi-GRU). These models utilize CNN to extract spatial correlations, LSTM or GRU to capture temporal dependencies, and Bi-LSTM or Bi-GRU to incorporate bidirectional temporal features from past and future data. The experiment is conducted on the GEANT dataset, with performance measured using MSE, RMSE, and MAE. The results showed that the CNN–LSTM–Bi-LSTM model achieved an 8% reduction in MSE at 20 epochs and 96% at 100 epochs, and the CNN–GRU–Bi-GRU model showed a 14% reduction in MSE for 20 epochs compared to the LSTM-TMP model. Compared to the DeepTM model the CNN–LSTM–Bi-LSTM model improved by 6% reduction in MSE and the CNN–GRU–Bi-GRU by 13% for 20 epochs. Both models show a 96% reduction in MSE for 100 epochs over the ST-DNN model.

Zheng et al. [20] introduced a flow-by-flow (FBF) TMP method to remove the limitation of entire matrix (EM) methods. The EM models are only confined to a specific network topology and are difficult to apply to other networks. Instead, FBF utilizes the intra-flow temporal correlations. To generate samples for the prediction models, a sliding-window-based method is implemented. To evaluate the FBF method for TMP, three popular backbone models, namely, LSTM, GRU, and Transformer, are used. The FBF versions of LSTM–GRU and Transformer are compared to the EM versions of the same models. The experiments are conducted on the Abilene and GEANT datasets. The results show that on the Abilene dataset, the LSTM–FBF reduced the metrics by 74.75% (MSE) and 56.13% (MAE) when compared to LSTM–EM, by 66.57% and 48.32% for the GRU–FBF model, and by 25.67% and 17.07% for Transformer–FBF, respectively. On the GEANT dataset, LSTM–FBF reduced the metrics by 69.27% (MSE) and 63.09% (MAE) when compared to LSTM–EM, by 60.72% and 61.94% for the GRU–FBF model, and by 44.54% and 48.52% for the Transformer–FBF model, respectively.

Etengu et al. [21] proposed a deep ensemble learning model for network TMP in hybrid SDN/OSPF to enhance traffic engineering and resource management. The framework used an LSTM-based RNN architecture combined with three dimensionality reduction algorithms, namely, canonical correlation analysis (CCA), PCA, and independent component analysis (ICA), for feature extraction. A multi-objective genetic algorithm (MO-GA) optimizes the ensemble by dynamically adjusting connection weights, replacing manual trial-and-error methods. Experiments were conducted on the GEANT dataset. The Var-WAE MO-GA–LSTM model is compared to the GRU-based counterpart and

single baseline architectures. The model achieved a 96.9% accuracy rate, outperforming the baseline CCA–LSTM by a 9.12% increase, Fixed-WAE ensemble by a 29.03% increase, and consensus MO-GA ensemble by a 25.36% increase. Compared to the GRU-based counterpart, it estimated 9.12% increase using CCA–LSTM as a proxy. Compared to baseline architectures, the model likely had lower RMSE and higher efficiency.

Zhang et al. [22] integrated the Prophet Framework, a novel TM prediction solution for TE in wide area networks. The problem addressed is the need for accurate TM predictions to optimize network routing and reduce link congestion, which is challenging due to dynamic traffic patterns. Prophet solves this by preserving the critical property—the ratio among TM elements—that maintains TE performance even if the TM scales. The method has two main components: matrix normalization, which scales all TM elements by the largest value to preserve inter-element ratios, and a traffic engineering (TE)-centric angle loss function, which minimizes the angular difference between the predicted and true TMs. It uses a GRU model with historical TMs as input. Prophet was tested on Abilene and CERNET datasets. TE performance was evaluated using the performance ratio (PR), comparing TE results from predicted versus real TMs. Compared to baselines such as Informer, ACRNN, Equal-Cost Multi-Path (ECMP), and Oblivious Routing (OR), Prophet significantly outperformed, achieving 45.4% better PR for link-level TE and up to a 52.8% improvement for path-level TE across both datasets. It also demonstrated faster computation than ACRNN, reducing training time by about two-thirds on Abilene.

Zheng et al. [23] proposed an end-to-end deep learning framework, 3DUNet LSTM2D, for TMP with randomly missing values. For the completion of missing values, a masked matrix modeling method based on self-supervised learning inspired from the matrix image modeling is introduced. This framework performs the two tasks simultaneously through joint learning. A 3D-UNet architecture that can exploit multi-scale spatio-temporal correlations in a TM sequence as the completion module. An LSTM2D architecture is employed as the prediction module to take advantage of spatio-temporal dependencies. The framework is compared to eight baseline models, namely, zero-filling, mean-filling,

Table 1
Summary of state-of-the-art works

Research work	Addressed issue	Technique/algorithm used	Performance metrics	Results
[13]	Scalability issues in TE estimation for IP networks	PCAOM with Moore–Penrose inverse	SRE, TRE	Reduced SRE, TRE compared to tomogravity with better performance on the noisy data.
[14]	TM prediction and estimation in data center networks	DBN with logistic regression for prediction; DBN with iterative proportional fitting for estimation	Error improvement ratio, bias, standard deviation	DBN significantly improved accuracy in both datasets, with low bias but slightly higher variance than SRMF in prediction.
[15]	Challenges in monitoring large-scale IP backbone networks	DBN with sliding window for prediction; DBN with iterative proportional fitting for estimation	Bias, standard deviation	DBN outperformed with lower bias and standard deviation, particularly for short-term traffic variations.
[16]	End-to-end traffic estimation in high-load computer networks	ConvNet with L2AMs for graph embedding	SRE, TRE	Lower SRE and TRE, with better tracking of traffic fluctuations.
[17]	Estimating traffic matrices with complex spatial-temporal dependencies	Multi-view subspace learning	RTE, RSE	CCA reduced temporal and spatial errors with near-zero bias and variance, and faster execution.
[18]	Accurate TM estimation in cloud networks	CNN	SRE, TRE, bias, standard deviation	R-CNTME outperformed CNTME, with improved anomaly detection.
[19]	Accurate TM prediction for network management and planning	LSTM, GRU	MSE, RMSE, MAE	Both hybrid models improved prediction accuracy, with the CNN–GRU–Bi-GRU model showing slightly better performance.
[20]	Low cost and high adaptability of traditional TM prediction	Flow-by-flow with Times Net model	MSE, MAE	Achieved high accuracy on GEANT and Abilene datasets.
[21]	Deep ensemble learning model for network TM prediction	MO-GA–LSTM ensemble using CCA, PCA, ICA and LSTM	Accuracy rate, RMSE	Var-WAE MO-GA–LSTM ensemble outperformed baselines, and achieved higher accuracy and consistency.
[22]	TM estimation focused TE needs	Prophet	PR	Prophet improved the performance of link-level TE and path-level TE, compared to existing solutions.
[23]	Predicting network traffic with incomplete data and high accuracy	3DUNet LSTM2D with MMM	RMSE, R^2	3DUNet LSTM2D outperformed baseline completion and prediction models, achieving lower RMSE and higher R^2 across various missing rates.
[24]	Network traffic prediction based on combination model	ARIMA–IPSO–Bi-LSTM	RMSE, MAE, MAPE, RRMSE, R^2	ARIMA–IPSO–Bi-LSTM outperformed ARIMA, Bi-LSTM, achieving lower errors and higher R^2 for single and multi-step predictions.

KNN (k = 2), MC-NMF, LMaFit, IALM-MC, SRCNN, and GCRINT, and three prediction models, namely, LSTM2D, LSTNet, and MTGNN. The experiments are conducted on the GEANT and Abilene datasets with missing rates from 10% to 90%. The results show 96.9% reduction in RMSE for Abilene and 62.6% for GEANT. R^2 increases at 13.5% for Abilene and at 20% for GEANT.

Li and Tian [24] proposed a network traffic prediction model using a combination model integrating the autoregressive integrated moving average (ARIMA) with an improved particle swarm optimization (IPSO) optimized Bi-LSTM network. This model addresses the challenge of predicting network traffic with both linear and nonlinear characteristics. The model employs ARIMA to model the linear components of traffic data, while the residual nonlinear patterns are captured by an IPSO optimized Bi-LSTM, where IPSO fine-tunes hyperparameters such as learning rate and hidden layer nodes for improved accuracy. The experiment is conducted on the two real world datasets, namely, Dalian city and European city datasets, with varying fluctuation patterns. The ARIMA–IPSO–Bi-LSTM model is compared to the ARIMA [25] and Bi-LSTM. The results show that for one-step prediction, it reduces RMSE by over 50% and increases R^2 by up to 13%. In two-step prediction, it reduces errors such as RMSE and MAPE by 70%–80%. Even in three-step prediction, the model sustains over 60% error reduction with R^2 consistently above 0.98, showing that the model works well for both short-term and long-term traffic forecasts.

Although the existing studies have explored TM prediction and estimation using deep learning and hybrid approaches, several limitations exist. Many works focus on traffic estimation on single topology by limiting their applicability across the diverse network topologies. Some approaches focus on spatial dependencies, which increases the computation cost. Only few studies analyze the attention-based model for TMP. To fill these gaps the proposed model is introduced, where it purely focuses on temporal dependencies to predict the future TMs in SDN networks.

Table 1 shows a summary of state-of-the-art works.

3. Proposed Methodology

3.1. Problem formulation

Let us assume that a network with k nodes consist of k^2 end-to-end flows. The traffic among each flow is shown using a TM V. For any timestep t, the TM is represented as V_t. To predict the future TM F_t, the sequence of past TMs are considered for a given time horizon. The prediction of TM at a time step t by considering the sequence of M past TMs is shown in Equation (13):

$$F_t = P(V_{t-1}, V_{t-2}, \ldots\ldots, V_{t-M}). \tag{13}$$

Here, the function P represents the prediction model that takes a sequence of M past TMs to predict the F_t of the time step t.

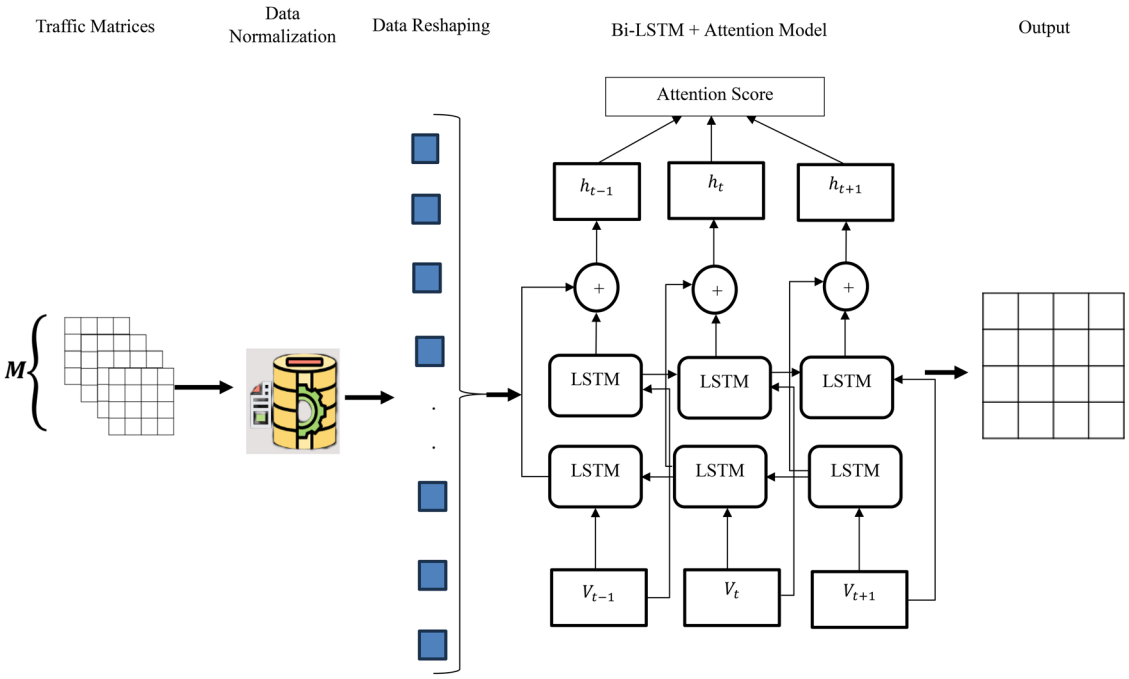
3.2. Proposed work

A Bi-LSTM model with attention mechanism is proposed for more accurate TMP in SDN. The architecture diagram of the proposed model is given in Figure 5. The model considers V as input, which contains the past TMs of varying time steps. Then, V is normalized using the MinMax normalization technique, where the values are scaled between 0 and 1. After normalization, V is reshaped to align with the proposed model's input structure.

Later, it passes through Bi-LSTM model that contains two layers, which are forward LSTM and backward LSTM. At each timestep, both layers of LSTM produce a hidden state as output. These hidden states obtained from both layers will be concatenated as $[\overrightarrow{h_t}, \overleftarrow{h_t}]$ and give the output from the Bi-LSTM layer as H_t.

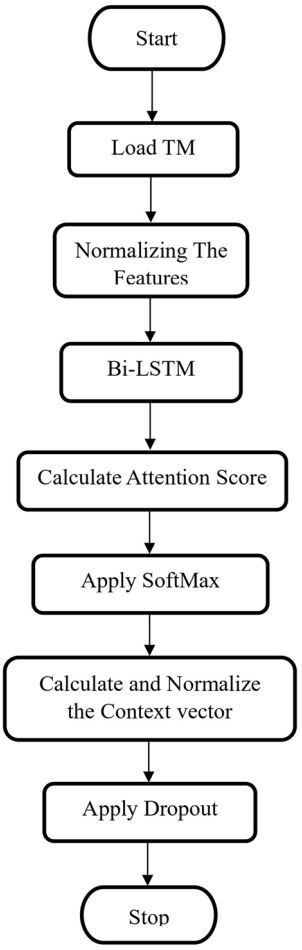
Next, these hidden states pass through the attention mechanism, where it calculates the score for every H_t. Following that, these scores undergo the softmax normalization technique, which gives attention scores. Using this attention score and hidden states, the model calculates the context vector that is shown in Figure 6, which then undergoes

Figure 5
Block diagram of the Bi-LSTM model with attention mechanism



softmax normalization. Thereafter, to prevent overfitting, a dropout layer is applied to the context vector, which then passes through a final fully connected layer. This layer predicts the TM for the next time step.

Figure 6
Workflow of the Bi-LSTM model with attention mechanism



Notations and their descriptions are listed in Table 2.

Table 2
List of notations

Notation	Description
V	Input traffic matrices
M	Total number of time steps
N	Total number of nodes
X_{norm}	Normalized values of a time step
$\hat{X}$	Normalized traffic matrices
H_t	Hidden state for t time step
α	Attention score
$\hat{\alpha}$	Normalized attention score
c	Context vector
$\hat{C}$	Normalized context vector
C_{drop}	Context vector after dropout
$\hat{Y}$	Predicted traffic matrix

3.2.1. MinMax normalization

The MinMax normalization technique is used to normalize the traffic demand values in the TMs. The traffic demand values are normalized to scale it between 0 and 1. Such scaling provides a better input to the model for future TMP. A detailed explanation is given in Algorithm 1. The algorithm considers V as input and processes it accordingly to generate $\hat{X}$. The formula for the MinMax normalization is shown in Equation (14).

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}. \tag{14}$$

Algorithm 1: MinMax normalization

Input: V

1: **for** i ← 1 to M **do**
2: **for** j ← 1 to N **do**
3: $\boldsymbol{X_{norm}} \leftarrow 0$
4: $\boldsymbol{Max_{val}} \leftarrow \boldsymbol{max(j)}$
5: $\boldsymbol{Min_{val}} \leftarrow \boldsymbol{min(j)}$
6: $\boldsymbol{X_{norm}[i,j]} \leftarrow \frac{\boldsymbol{X[i,j] - Min_{val}}}{\boldsymbol{Max_{val} - Min_{val}}}$
7: $\hat{\boldsymbol{X}} \leftarrow \boldsymbol{X_{norm}}$
8: **end for**
9: **end for**
10: return $\hat{\boldsymbol{X}}$

Output: $\hat{\boldsymbol{X}}$

In line 1, the algorithm iterates through M timesteps, and in line 2, it iterates through N nodes, where these loops help in iterating through the dataset. Now in this loop, line 3 is initializing zero to $\boldsymbol{X_{norm}}$. Lines 4 and 5 search for the maximum and minimum values in every time step. Line 6 calculates the MinMax normalization for all values, which are stored in $\boldsymbol{X_{norm}}$. In line 7, all values $\boldsymbol{X_{norm}}$ are stored in $\hat{X}$. Line 10 is returning the normalized dataset as $\hat{X}$. The final output from this algorithm is the complete normalized dataset, with values ranging from 0 to 1.

3.2.2. Bi-LSTM with attention mechanism

Bi-LSTM processes the input in forward and backward directions, which capture the past and future temporal dependencies. This is very useful in network traffic patterns because it can cover dependencies over multiple time steps and all time steps did not contribute equally to the prediction. Therefore, the attention mechanism is used where it helps in assigning different weights dynamically for each timestep by giving most attention to the important parts of the sequence. The type attention mechanism used in this algorithm is the learnable additive attention mechanism. It works by computing the relevance score for each Bi-LSTM hidden state using a fully connected layer, followed by a softmax function to obtain normalized attention weights. Owing to this mechanism, the model achieves better prediction. The proposed Bi-LSTM with attention mechanism is given in Algorithm 2.

The formula for the attention weights is shown in Equation (15):

$$z = \frac{e^{attention\ weight}}{\sum e^{attention\ weight}}. \tag{15}$$

In lines 2 and 3, the algorithm is iterating through M time steps and N nodes. In lines 3-4 within the loop, we are calculating hidden state $\boldsymbol{H_t}$ for every timestep T to calculate the overall hidden state. It

Algorithm 2: Bi-LSTM with attention mechanism

Input: $\hat{X}$

1: **for** i← 1 to M **do**
2: **for** j ← 1 to N **do**
3: Calculate all the Bi-LSTM gates and give Hidden State (H_t)
4: $$H_t \leftarrow [\overrightarrow{h_t}, \overleftarrow{h_t}]$$
5: Compute attention score (α)
6: $$\alpha \leftarrow tanh(H_t.W_\alpha)$$
7: **end for**
8: **end for**
9: Applying softmax to the attention score
10: $$\hat{\alpha} \leftarrow \frac{e^{\alpha}}{\sum e^{\alpha}}$$
11: **for** i = 1 to T **do**
12: Calculate Context Vector(c):
13: $$c \leftarrow \sum \alpha_T.H_T$$
14: **end for**
15: Apply Layer Normalization $(\hat{c})$:
16: $$\hat{c} \leftarrow LayerNorm(c)$$
17: Apply Dropout (c_{drop})
18: $$c_{drop} \leftarrow dropout(\hat{c})$$
19: Compute Final Predictions:
20: $$\hat{Y} \leftarrow W_{fc} * c_{drop} + bias$$
21: Return $\hat{Y}$

Output: $\hat{Y}$

undergoes two layers, namely, the forward LSTM and backward LSTM. In lines 5-6, the calculation of attention score α for each time step T is conducted. Lines 9-10 are about applying softmax to the attention score, which normalizes the attention score. Lines 11-14 involve calculating the context vector for every time step. Lines 15-16 are normalizing the context vector. Lines 17-18 are about applying dropout for the context vector. Lines 19-20 are calculating the final predictions using the dropout context vector and the weighted matrix. Line 21 is returning the final prediction of TM for the next time step.

3.2.3. Mathematical example of the Bi-LSTM model with attention mechanism for accurate TMPs

1) Step 1: Considering a TM with end-to-end flows

A TM of 5 nodes and 5 links with 6 timesteps is considered for sample evaluation. The considered matrix is presented as a table for better visualization and understanding. Table 3 depicts the values of end-to-end flows between nodes with respect to each timestep.

2) Step 2: Normalization of traffic values between node

The traffic volumes are normalized using MinMax normalization, which scales the values to the range of 0–1. The computed normalized values are shown in Table 4.

Table 3
Example topology

Timesteps	Link 1-2	Link 2-3	Link 3-4	Link 4-5	Link 5-1
Timestep1	19.879	19.879	29.810	11.117	21.190
Timestep2	20.247	20.247	26.967	11.239	22.465
Timestep3	21.456	21.456	27.413	10.832	23.721
Timestep4	22.072	22.072	31.073	13.372	23.578
Timestep5	20.577	20.577	27.262	11.009	20.366
Timestep6	19.364	19.364	31.742	11.207	20.892

For timestep1
Link 1-2 Min = 27.978, Max = 40.929
Link 2-3 Min = 19.364, Max=22.072

$$Link\ 1-2 = \frac{36.398-27.978}{40.929-27.978} \approx 0.647$$

$$Link\ 2-3 = \frac{19.879-19.364}{22.072-19.364} \approx 0.190.$$

Table 4
Normalized values of the given TM

Timestep	Link 1-2	Link 2-3	Link 3-4	Link 4-5	Link 5-1
1	0.650	0.650	0.595	0.112	0.245
2	1.000	1.000	0.000	0.160	0.626
3	0.648	0.648	0.094	0.000	1.000
4	0.707	0.707	0.861	1.000	0.958
5	0.631	0.631	0.062	0.070	0.000
6	0.000	0.000	1.000	0.147	0.156

Similarly, calculate for all nodes in all timesteps. Table 4 shows the all-normalized values for all nodes in the given topology.

3) Step 3: Reshaping of data to fit into the model

Reshaping the traffic data according to the input that has to be given to the model. The reshaping follows as

$$R \leftarrow 1batch,\ 6timesteps,\ 5nodes.$$

4) Step 4: Providing the reshaped values as input to the model

The normalized values pass through the Bi-LSTM model, where it passes through two layers, namely, forward pass and backward pass. Here, the concatenated hidden states of forward pass and backward pass are obtained as a combined hidden state, which serves as output.

For timestep1

Hidden size = 2

Compute gates for the timestep1

$$Input\ x_t = [0.650,\ 0.190,\ 0.595,\ 0.112,\ 0.245]$$

$$Previous\ Hidden\ State\ h_0 = [0, 0]$$

$$Previous\ cell\ state\ c_0 = [0, 0].$$

Compute all gates

Forget gate (f_t):

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) = \sigma([0.302 + 0.1, 0.417 + 0.2]) = [0.598, 0.653].$$

Likewise compute all gates

Input gate (i_t) & cell state ($\widetilde{C}_t$):

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) = [0.4, 0.5]$$

$$\widetilde{C}_t = tanh(W_c * [h_{t-1}, x_t] + b_c) = [0.2, 0.3].$$

Cell state (C_t):

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t = [0.08, 0.15].$$

Output gate (o_t):

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) = [0.6, 0.7].$$

Hidden state ($h_t^{forward}$) :

$$h_t^{forward} = o_t \odot tanh(C_t) = [0.2, 0.3].$$

$$Forward\ Pass = [0.2, 0.3]$$

Likewise, calculate the backward pass

$$Backward\ Pass = [0.1, 0.2]$$

$$Bi - LSTM(T1) = concat([0.2, 0.3], [0.1, 0.2]) = [0.2, 0.3, 0.1, 0.2].$$

Similarly, calculate for all timesteps. Table 5 shows all hidden states for all timesteps.

Table 5
Hidden states for all timesteps

Timestep	Hidden state
1	[0.2,0.3,0.1,0.2]
2	[0.1,0.4,0.3,0.3]
3	[0.5,0.5,0.4,0.2]
4	[0.6,0.7,0.5,0.1]
5	[0.3,0.2,0.6,0.6]
6	[0.4,0.1,0.7,0.4]

5) Step 5: Calculation of attention score

The hidden state values are passed through an attention mechanism. Here, the attention scores will be calculated using hidden states.

The example of the calculation is shown below:

The weights and bias are assumed as below:

W = [0.2.0.4,0.3,0.1], b =0.05

For timestep1

$$\alpha = 0.2\ x\ 0.2\ +\ 0.4\ x\ 0.3\ +\ 0.3\ x\ 0.1\ +\ 0.1\ x\ 0.2\ =\ 0.254.$$

Similarly, calculate for all timesteps.

The calculated attention scores is given below:

$$\alpha = [0.254, 0.388, 0.454, 0.544, 0.405, 0.397].$$

6) Step 6: Applying softmax over the attention scores

The softmax activation function is applied on the attention scores. The sample calculation is shown below:

For timestep1,

$$e^t = e^{0.254} \approx 1.289.$$

Likewise, the calculation of all timesteps is

$$e^\alpha = [1.289,\ 1.475,\ 1.575,\ 1.723,\ 1.5,\ 1.487]$$

$$\sum e^\alpha \approx 9.049.$$

Now, attention weights are

$$\widehat{\alpha_1} = \frac{1.289}{9.049} \approx 0.1424.$$

Likewise, all attention weights are

$$\widehat{\alpha} = [0.1424,\ 0.1630,\ 0.1740,\ 0.1904,\ 0.1658,\ 0.1643].$$

7) Step 7: Calculation of the context vector for the attention weights

The calculation of the context vector for attention weights is shown below:

For timestep1,

$$\widehat{\alpha} = 0.1424$$

$$H_t = [0.2,\ 0.3,\ 0.1,\ 0.2]$$

$$c_1 = 0.1424 * [0.2,\ 0.3,\ 0.1,\ 0.2] = 0.3614.$$

Likewise, the complete context vector is calculated and resultant value is shown below:

$$c = [0.3614,\ 0.3778,\ 0.4424,\ 0.2964].$$

8) Step 8: Normalization of the context vector

The steps to normalize context vector is shown below:

For the context vector(c)

Mean = 0.2964

Standard deviation = 0.146

$$\widehat{c_1} = \frac{0.3614 - 0.2964}{0.146} = 0.445.$$

Calculate for all values; the normalized context vector is

$$\hat{c} = [0.445,\ 0.558,\ 1.000,\ 0.000].$$

9) Step 9: Applying the dropout layer

To prevent overfitting, dropout is applied to the context vector. The dropout value for the current observation is "0.25." After applying dropout to the context vector, c_{drop} is obtained as

$$c_{drop} = [0.445,\ 0.000,\ 1.000,\ 0.000].$$

10) Step 10: Final predictions

The vector passes through the fully connected linear layer, where it calculates the final predictions. The calculation is shown below:

Let us assume w = [0.2,0.3,0.2,0.2]

$$\widehat{Y}_t = 0.445 * 0.2 + 0 * 0.3 + 1 * 0.2 + 0 * 0.2 = 0.28.$$

Hence, $\hat{Y}$ denotes predicted values, which are in the range of the previous values.

4. Experimentation

This section shows the results of the proposed model. The proposed Bi-LSTM with attention mechanism is compared with the existing Bi-LSTM with Adam model. The performance metrics MSE and MAE are considered to evaluate the proposed model.

4.1. Environmental setup

The hardware and software specifications used for the proposed work are given in Table 6.

Table 6
Hardware and software specifications

Component	Specification
Processor	Intel i5-1235U 1.30 GHz
RAM	16 GB
Operating system	Windows 10
Platform	Jupyter Notebook
Programming language	Python 3.9
Libraries used	numpy, torch, pandas, sklearn
Storage device	512 GB SSD

4.2. Dataset information

Four different datasets, namely, Abilene, GEANT, Germany50, and Nobel-Germany, are considered. The granularity, horizon, and number of matrices of each topology are given in Table 7. Granularity represents the time interval between measurements of the matrix. Horizon represents the time duration that TMs are considered.

Table 7
Dataset information

Network	Granularity	Horizon	Number of Matrices
Abilene	5 min	6 months	48,096
GEANT	15 min	4 months	11,460
Nobel-Germany	5 min	1 day	288
Germany50	5 min	1 day	288

The datasets were divided into training, testing, and validation sets using a chronological holdout strategy, where 70% is the training data, 10% is for validation, and 20% is for the testing of the data. The 70–10–20 split was used because allocating 70% of the data for training allows the model to learn long-term dependencies present in the TMs, while using 20% for testing provides sufficient unseen data for robust generalization assessment. The validation set is used for hyperparameter tuning, ensuring that the test results remain unbiased. For this model, the 70–10–20 split showed more stable validation behavior across all datasets with different granularities and traffic horizons compared to alternative splits such as 80–10–10.

The input sequence length or look-back window was set to 24 timesteps for all datasets. This value was selected based on a trial-and-error method, where multiple window sizes such as 6, 12, 24 and 48 timesteps were evaluated. Among these, the 24-timestep window consistently provided the best predictive results and offered sufficient amount of data to capture temporal dependencies without increasing computation cost. For fair comparison across all topologies, the same window size was applied to all datasets.

Before model training, all datasets will undergo MinMax normalization, which is a preprocessing technique where the values are scaled between 0 and 1.

4.3. Performance metrics

The proposed model is evaluated using performance metrics such as MSE and MAE. These metrics are essential to understand how well the model is performing and what is the error percent of the model and say how accurately the model is predicting the future matrices.

4.3.1. Mean squared error

Mean squared error (MSE) is a metric that is used to predict the accuracy of the given model. This metric calculates the average of the squared difference between the predicted values and the actual values. The error is calculated using Equation (16).

$$MSE = \frac{1}{n}.\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_l\right)^2, \tag{16}$$

where

N represents the number of nodes,
Y_i represents the actual values,
$\widehat{Y}_l$ represents the predicted values.

4.3.2. Mean absolute error

Mean absolute error (MAE) is a metric that measures the average of absolute difference between the predicted values and the actual values. The error is calculated using Equation (17).

$$MAE = \frac{1}{n}.\sum_{i=1}^{n}\left|Y_i - \widehat{Y}_l\right|, \tag{17}$$

where

N represents the number of nodes,
Y_i represents the actual values,
$\widehat{Y}_l$ represents the predicted values.

4.3.3. Coefficient of determination

Coefficient of determination (R^2) is a metric that measures the square of the sample correlation coefficient (R) between the observed outcomes and the observed predictor values. The error is calculated using Equation (18).

$$R^2 = 1 - \frac{\sum_i^n \left(Y_i - \widehat{Y}_l\right)^2}{\sum_i^n \left(Y_i - \overline{Y}\right)^2}, \tag{18}$$

where

N represents the number of nodes,
Y_i represents the actual values,
$\widehat{Y}_l$ represents the predicted values,
$\overline{Y}$ represents the mean of the observed data.

4.3.4. Average inference time

Average inference time is a metric that measures the time required by the trained model to generate predictions during forward pass, computed over multiple inference runs without including training and backpropagation. The calculation of the average inference time is given in Equation (19).

$$Average\ Inference\ Time = \frac{1}{N}\sum_{i=1}^{N} t_i, \tag{19}$$

where

t_i represents time taken for the i^{th} forward inference,
N represents number of inference runs.

4.4. Results and discussion

This section discusses the results that are predicted by the proposed model and discusses how well the proposed model is performing compared to the existing model, which is Bi-LSTM with Adam.

Bi-LSTM with Adam outperforms the plain Bi-LSTM model; therefore, Bi-LSTM with Adam is considered the baseline model.

4.4.1. Hyperparameters for Bi-LSTM with attention mechanism for different datasets

This section shows the hyperparameters at which the proposed model performs best for the different datasets. Tables 8, 9, and 10 show the tuned hyperparameter settings used for the Abilene, GEANT, Germany50, and Nobel-Germany topologies.

Table 8
Hyperparameters for the Abilene and GEANT topologies

Hyperparameter	Value
Hidden dimension	256
Number of layers	3
Dropout	0.3
Batch size	64
No. of epochs	300

Table 9
Hyperparameter for the Germany50 topology

Hyperparameter	Value
Hidden dimension	512
Number of layers	3
Dropout	0.3
Batch size	64
No. of epochs	100

Table 10
Hyperparameter for the Nobel-Germany topology

Hyperparameter	Value
Hidden dimension	256
Number of layers	3
Dropout	0.3
Batch size	64
No. of epochs	100

These hyperparameters of the proposed model were selected through manual empirical tuning based on validation performance. Different parameter combinations were evaluated, and the lowest error parameters were selected for each of the dataset.

4.4.2. Comparison of MSE

The MSE of the proposed model is compared with that of the Bi-LSTM with Adam model, which is shown in Figure 7. The proposed Bi-LSTM model with attention mechanism predicts the matrices with minimal error compared to the Bi-LSTM with Adam model.

Figure 7 shows the comparison of the MSE values for different topologies, where it shows that for the Nobel-Germany, Germany50, and GEANT topologies, it works well with the proposed model, whereas for Abilene, it shows similar values with the existing model.

4.4.3. Comparison of MAE

The MAE of the proposed model is compared with that of the Bi-LSTM with Adam model, which is shown in Figure 8. The proposed

Bi-LSTM model with attention mechanism predicts the matrices with minimal errors compared to the Bi-LSTM with Adam model, which indicates that the proposed model is working effectively.

Figure 7
Comparison of MSE for different topologies

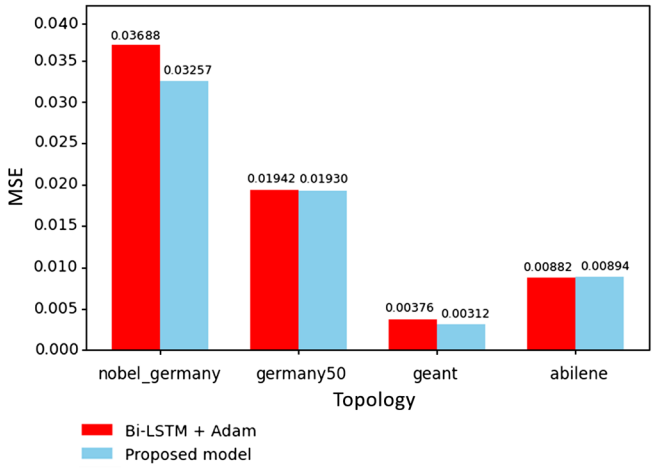


Figure 8
Comparison of MAE for different topologies

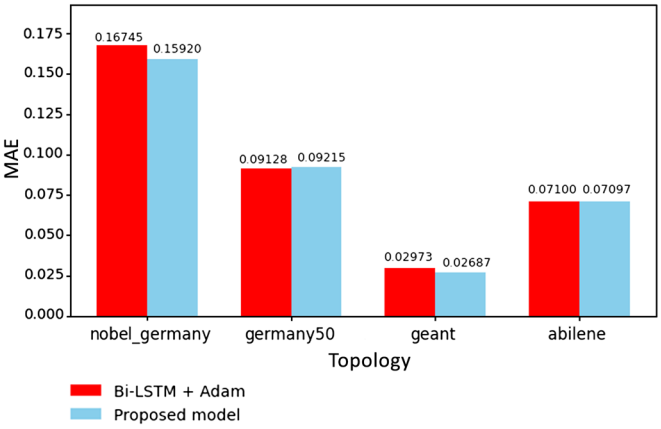


Figure 8 shows the comparison of MAE values for the different topologies, where it shows that Nobel-Germany, GEANT, and Abilene perform well with these topologies, whereas for Germany50, it shows similar results with the previous models.

4.4.4. Comparison of R^2

The R^2 of the proposed model is compared with the Bi-LSTM with Adam model, which is shown in Figure 9. The proposed Bi-LSTM model with attention mechanism predicts the matrices with minimal error compared to the Bi-LSTM with Adam model.

Figure 9 shows the comparison of R^2 values for different topologies, where it shows that for the Nobel-Germany, Abilene, and GEANT topologies, it works well with the proposed model, whereas for Germany50, it shows similar values with the existing model.

4.4.5. Comparison of average inference time

The average inference time of the proposed model is compared with that of the Bi-LSTM with Adam model, which is shown in Figure 10. The proposed Bi-LSTM model with attention mechanism predicts the matrices with minimal errors compared to the Bi-LSTM with Adam model.

Figure 9
Comparison of R^2 for different topologies

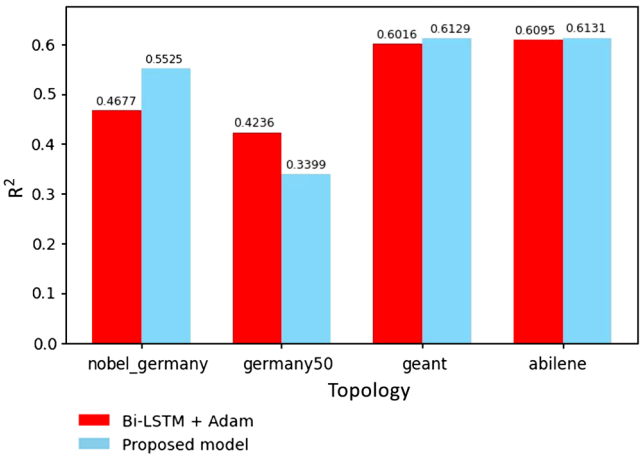


Figure 10
Comparison of inference time for different topologies

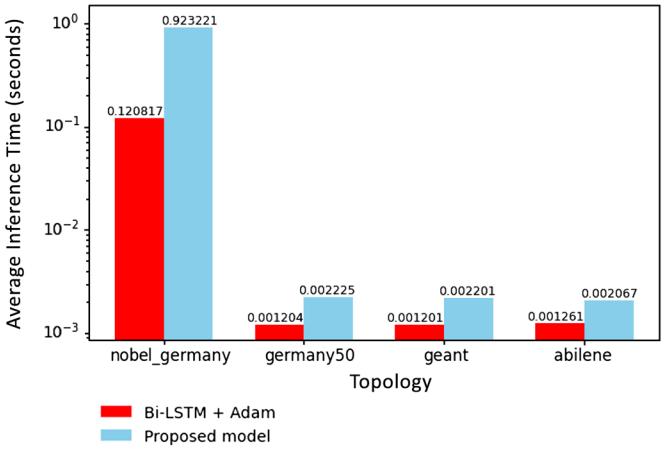


Figure 10 shows that the inference time of the proposed model is higher for all topologies. The attention mechanism increases the complexity of the model, leading to additional computation during prediction. However, the existing Bi-LSTM with Adam model is lightweight compared to the proposed model, resulting in faster predictions.

The differences in granularities of the topologies explains the variation in prediction performance across the different topologies. The datasets GEANT and Nobel-Germany contain more fluctuating and irregular traffic patterns. In addition, the granularity is more like 5 and 15 min. Hence, the data show the differences, such as some sudden spikes and dropouts, and that is where the attention mechanism works perfectly, which shows better results than the baseline model. Although Germany50 shares the same granularity as Nobel-Germany, the traffic data in Germany50 exhibit comparatively smoother and more temporally correlated behavior. As a result, consecutive traffic matrices show similar patterns, allowing the baseline Bi-LSTM model to effectively learn temporal dependencies. Here, the attention mechanism observes all timesteps with the same importance, so the result is similar to the baseline model. Likewise, in case of Abilene the data is stable and predictable so it exhibits lower sudden spikes and less irregularity that is why it shows similar results when compared with the baseline model.

5. Conclusion

Predicting TMs remains one of the major challenges due to the dynamic and complex traffic patterns. This paper addresses this challenge by focusing on forecasting future TMs for improved network management and congestion control. The results showed that Bi-LSTM with attention mechanism could predict the future TMs more efficiently by considering historical TMs. The model performs well across all considered datasets by achieving low MSE and MAE values. The results of the proposed model help network administrators make better decisions to prevent overloading and congestion control, to reduce delays, and to improve overall network efficiency. In practical SDN environments, the predicted TMs can be integrated into the SDN controller's decision loop to enable proactive traffic engineering and effective congestion control. However, the effectiveness of the deployment may depend on the scalability and resource constraints of the SDN controller. This work is limited to the SDN topologies and focuses on predicting the future TMs for "n" timesteps. The proposed model primarily captures temporal dependencies in traffic patterns but does not model spatial dependencies such as graphs networks. The model can be further improved by including advanced hybrid architectures such as combining Bi-LSTM with Transformers-based models to enhance both prediction accuracy and adaptability. Overall, the proposed model offers an effective way for the prediction of future TMs.

Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable comments.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in SNDlib at https://sndlib.put.poznan.pl/home.action.

Author Contribution Statement

Prabu. U.: Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data curation, Writing – original draft, Visualization. **Prabhala Harichandana:** Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data curation, Writing – original draft, Visualization. **Manepalli Baby Kavyasri:** Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data curation, Writing – original draft, Visualization. **Uppala Bhargavi:** Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data curation, Writing – original draft, Visualization. **Geetha. V.:** Formal analysis, Writing – review & editing, Supervision, Project administration.

References

[1] Hu, H., Ke, F., Qin, M. J., & Lee, Y. L. (2025). Traffic matrix prediction based on multilevel discrete wavelet transform network and LSTM. *Transactions on Emerging Telecommunications Technologies*, *36*(5), e70159. https://doi.org/10.1002/ett.70159

[2] Priyadarsini, M., & Bera, P. (2021). Software defined networking architecture, traffic management, security, and placement: A survey. *Computer Networks*, *192*, 108047. https://doi.org/10.1016/j.comnet.2021.108047

[3] Islam, M. S., Al-Mukhtar, M., Khan, M. R. K., & Hossain, M. (2023). A survey on SDN and SDCN traffic measurement: Existing approaches and research challenges. *Eng*, *4*(2), 1071–1115. https://doi.org/10.3390/eng4020063

[4] Kumar, A., Singh, N. H., Namasudra, S., Crespo, R. G., & Moparthi, N. R. (2024). Traffic matrix estimation using matrix-CUR decomposition. *Computer Communications*, *217*, 200–207. https://doi.org/10.1016/j.comcom.2024.02.002

[5] Wassie, G., Ding, J., & Wondie, Y. (2023). Traffic prediction in SDN for explainable QoS using deep learning approach. *Scientific Reports*, *13*(1), 20607. https://doi.org/10.1038/s41598-023-46471-8

[6] Wang, X., Sun, Y., Wang, X., Wang, E., & Sun, Z. (2023). Traffic matrix prediction in SDN based on spatial-temporal residual graph convolutional network. In *Chinese Control and Decision Conference*, 3382–3387. https://doi.org/10.1109/CCDC58219.2023.10327651

[7] Wen, J., & Wang, Z. (2022). Bi-directional BILSTM-attention short-term load forecasting based on correlation weight. In *International Conference on Electronic Technology, Communication and Information*, 50–54. https://doi.org/10.1109/ICETCI55101.2022.9832041

[8] Jiang, W. (2022). Internet traffic matrix prediction with convolutional LSTM neural network. *Internet Technology Letters*, *5*(2), e322. https://doi.org/10.1002/itl2.322

[9] Tang, X. (2024). Optimizing LSTM, Bi-LSTM, and GRU models with SSA for daily electricity forecasting. In *International Conference on Intelligent Computing and Human-Computer Interaction*, 53–58. https://doi.org/10.1109/ICHCI63580.2024.10808092

[10] Wang, J., Wang, H., & Li, K. (2024). Weather prediction model based on CNN-BiLSTM-attention. In *International Conference on Intelligent Computing and Data Mining*, 1–5. https://doi.org/10.1109/ICDM63232.2024.10762629

[11] Méndez, M., Merayo, M. G., & Núñez, M. (2023). Long-term traffic flow forecasting using a hybrid CNN-BiLSTM model. *Engineering Applications of Artificial Intelligence*, *121*, 106041. https://doi.org/10.1016/j.engappai.2023.106041

[12] Xu, S., Kodialam, M., Lakshman, T. V., & Panwar, S. S. (2021). Learning based methods for traffic matrix estimation from link measurements. *IEEE Open Journal of the Communications Society*, *2*, 488–499. https://doi.org/10.1109/OJCOMS.2021.3062636

[13] Zhao, E., & Tan, L. (2015). A PCA based optimization approach for IP traffic matrix estimation. *Journal of Network and Computer Applications*, *57*, 12–20. https://doi.org/10.1016/j.jnca.2015.07.006

[14] Nie, L., Jiang, D., Guo, L., Yu, S., & Song, H. (2016). Traffic matrix prediction and estimation based on deep learning for data center networks. In *Globecom Workshops*, 1–6. https://doi.org/10.1109/GLOCOMW.2016.7849067

[15] Nie, L., Jiang, D., Guo, L., & Yu, S. (2016). Traffic matrix prediction and estimation based on deep learning in large-scale IP backbone networks. *Journal of Network and Computer Applications*, *76*, 16–22. https://doi.org/10.1016/j.jnca.2016.10.006

[16] Emami, M., Akbari, R., Javidan, R., & Zamani, A. (2019). A new approach for traffic matrix estimation in high load computer networks based on graph embedding and convolutional neural network. *Transactions on Emerging Telecommunications Technologies*, *30*(6), e3604. https://doi.org/10.1002/ett.3604

[17] Kumar, A., Vidyapu, S., Saradhi, V. V., & Tamarapalli, V. (2020). A multi-view subspace learning approach to internet traffic matrix estimation. *IEEE Transactions on Network and Service Management*, *17*(2), 1282–1293. https://doi.org/10.1109/TNSM.2020.2983329

[18] Memon, R. A., Qazi, S., & Khan, B. M. (2021). Design and implementation of a robust convolutional neural network-based traffic matrix estimator for cloud networks. *Wireless Communications and Mobile Computing*, *2021*(1), 1039613. https://doi.org/10.1155/2021/1039613

[19] Aloraifan, D., Ahmad, I., & Alrashed, E. (2021). Deep learning based network traffic matrix prediction. *International Journal of Intelligent Networks*, *2*, 46–56. https://doi.org/10.1016/j.ijin.2021.06.002

[20] Zheng, W., Li, Y., Hong, M., Fan, X., & Zhao, G. (2022). Flow-by-flow traffic matrix prediction methods: Achieving accurate, adaptable, low cost results. *Computer Communications*, *194*, 348–360. https://doi.org/10.1016/j.comcom.2022.07.052

[21] Etengu, R., Tan, S. C., Chuah, T. C., Lee, Y. L., & Galan-Jimenez, J. (2023). AI-assisted traffic matrix prediction using GA-enabled deep ensemble learning for hybrid SDN. *Computer Communications*, *203*, 298–311. https://doi.org/10.1016/j.comcom.2023.03.014

[22] Zhang, Y., Han, N., Zhu, T., Zhang, J., Ye, M., Dou, S., & Guo, Z. (2023). Prophet: Traffic engineering-centric traffic matrix prediction. *IEEE/ACM Transactions on Networking*, *32*(1), 822–832. https://doi.org/10.1109/TNET.2023.3293098

[23] Zheng, W., Li, Y., Hong, M., Zhao, G., & Fan, X. (2024). Network traffic matrix prediction with incomplete data via masked matrix modeling. *Information Sciences*, *657*, 119835. https://doi.org/10.1016/j.ins.2023.119835

[24] Li, G., & Tian, Z. (2024). A new method of network traffic prediction based on combination model. *Peer-to-Peer Networking and Applications*, *17*(3), 1075–1090. https://doi.org/10.1007/s12083-024-01630-0

[25] Bektemyssova, G., Ahmad, A. R., Mirzakulova, S., & Ibraeva, Z. (2022). Time series forecasting by the ARIMA method. *Scientific Journal of Astana IT University*, 14–23. https://doi.org/10.37943/HFCH4395

How to Cite: U., P., Harichandana, P., Kavyasri, M. B., Bhargavi, U., & V., G. (2026). An Enhanced Traffic Matrix Prediction in Software-Defined Networks Using Bi-LSTM with Attention Mechanism. *Artificial Intelligence and Applications*. https://doi.org/10.47852/bonviewAIA62026867