Received 31 October 2023; revised 29 December 2023; accepted 19 January 2024. Date of publication 26 January 2024; date of current version 12 February 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3358740

Generative Deep Learning Techniques for Traffic Matrix Estimation From Link Load Measurements

GRIGORIOS KAKKAVAS[1] (Graduate Student Member, IEEE), NIKOLAOS FRYGANIOTIS[1], VASILEIOS KARYOTIS[1,2] (Member, IEEE), AND SYMEON PAPAVASSILIOU[1] (Senior Member, IEEE)

[1]School of Electrical and Computer Engineering, National Technical University of Athens, 15780 Zografou, Greece

[2]Department of Informatics, Ionian University, Corfu 49132, Greece

CORRESPONDING AUTHOR: S. PAPAVASSILIOU (e-mail: papavass@mail.ntua.gr)

This work was supported by the National Technical University of Athens Research Committee Grant on “Network Management and Optimization” under Award 95028000. The publication of the article in OA mode was financially supported by HEAL-Link.

ABSTRACT Traffic matrices (TMs) contain crucial information for managing networks, optimizing traffic flow, and detecting anomalies. However, directly measuring traffic to construct a TM is resource-intensive and computationally expensive. A more practical approach involves estimating the TM from readily available link load measurements, which falls under the category of inferential network monitoring based on indirect measurements known as network tomography. This paper focuses on solving the problem of estimating the traffic matrix from link loads by utilizing deep generative models. The proposed models are trained using historical data—specifically, previously observed TMs—and are then leveraged to transform traffic matrix estimation (TME) into a simpler minimization problem in a lower-dimensional latent space. This transformed problem can be efficiently solved using a gradient-based optimizer. Our work aims to examine and test different model architectures and optimization approaches. The performance of the proposed methods is comparatively evaluated over a comprehensive set of suitable metrics on two publicly available datasets comprising actual traffic matrices obtained from real backbone networks. In addition, we compare our approach with a state-of-the-art method previously published in the literature.

INDEX TERMS Attention, generative deep learning, network tomography, traffic matrix estimation, variational autoencoder.

I. INTRODUCTION

THE *traffic matrix* (TM) measures the demand between all pairs of origin and destination entities within a network, typically representing nodes or sets of nodes. It captures the volume of traffic flowing between these specific origins and destinations, holding crucial information for network management, traffic engineering, and anomaly detection tasks. Traditional methods for constructing a TM involve direct measurements at network entry and exit points, realized by collecting packet traces, performing flow-level aggregation, or utilizing packet sampling [1]. Such approaches, however, entail significant administrative and computational costs. An alternative and more practical method is to estimate the TM using readily available link load measurements and leveraging existing routing information. The latter falls under the category of inferential network monitoring based on indirect measurements known as network tomography (NT) [2], [3].

Machine learning has found extensive applications in network management and monitoring due to its efficiency in modeling nonlinearities and capturing long-range spatiotemporal dependencies in network traffic. Notably, deep neural networks incorporating recurrent units, such as LSTMs or GRUs, have demonstrated effectiveness in TM prediction [4], [5]. This is an example of multivariate time series prediction that involves forecasting future network-wide traffic based on historical TM data, where the past traffic intensity values inform predictions for subsequent time steps. While the present work addresses a related problem, it focuses specifically on TM estimation from link load measurements based on a linear measurement model. In this context, the developed solution takes as input the measured vector of link counts, with each link potentially multiplexing multiple origin-destination (OD) flows, and outputs the corresponding traffic matrix. This formulation constitutes a *linear inverse problem*, involving the reconstruction of the unknown (vectorized) traffic matrix from an underdetermined system of noisy linear link measurements (i.e., there are many solutions that can explain the measurements).

Assuming a network with n nodes and m links, NT problems can be generally formulated as the following system of linear equations:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon}, \tag{1}$$

where $\boldsymbol{y} \in \mathbb{R}^{m\times 1}$ is the vector of observed measurements, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is the routing or measurement matrix representing the network topology, $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ is the vector of unknown parameters, and $\boldsymbol{\epsilon} \in \mathbb{R}^{m\times 1}$ is a noise/error term. The routing matrix is usually binary (i.e., a_{ij} is 0 or 1, indicating whether link i participates in routing path j or not), but its entries can also be probabilities in the case of multiple paths in a network due to load balancing considerations. The error term $\boldsymbol{\epsilon}$ is typically assumed to be zero or conveniently distributed (e.g., Gaussian, Poisson, binomial or multinomial). The goal is to estimate the unobserved vector $\boldsymbol{x}$ given the aforementioned linear model and the known vector of measurements $\boldsymbol{y}$. The challenge lies in the fact that the system of linear equations is heavily under-determined (i.e., the matrix $\boldsymbol{A}$ is not full-rank, and there are many solutions that fit the observations) since generally $m << n$.

Tailoring the above generic NT formulation to the *traffic matrix estimation* (TME) problem, the unknown traffic matrix can be represented as an $n \times n$ matrix with the element at row i and column j indicating the traffic volume between the origin node i and the destination node j. To further simplify the representation, the traffic matrix is vectorized, i.e., it is transformed into a p-dimensional vector $\boldsymbol{x}$, with p corresponding to the number of OD flows, namely n^2. Assuming a fixed routing configuration during the measurement period and a negligible error term $\boldsymbol{\epsilon}$, we can formulate the linear model $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{y}$ represents the m-dimensional vector of link counts and $\boldsymbol{A}$ is the $m \times p$ (or $m \times n^2$) routing matrix, whose element a_{ij} is assigned a value of 1 if OD flow j traverses link i, or 0 otherwise. The inherent ill-posed nature of the TME NT problem is typically tackled by employing statistical models and regularization techniques to introduce additional structural assumptions.

Generative deep learning has witnessed remarkable advancements and substantial progress in recent years, driving the field towards unprecedented capabilities and applications [6]. In general terms, generative deep learning focuses on developing models capable of generating new data that resembles samples from a given distribution. The fundamental idea is to train deep generative models to learn the underlying patterns and structure of the data, enabling them to generate novel samples with desirable characteristics. Deep generative models typically employ deep neural networks to capture the complex relationships and dependencies within the data [7]. These models aim to learn a latent representation or a compressed encoding of the data, which can then be used to generate new samples. They operate in an unsupervised learning setting, meaning they learn from unlabeled data without explicit target labels.

In contrast to classical approaches that involve sparsity assumptions or conditional independence conditions to solve linear inverse problems, a recent promising strategy leverages pre-trained deep generative models as priors [8]. The key premise of this methodology is the assumption that the unknown vector is in or near the range of the pre-trained deep generative model, i.e., there is a latent vector that can produce the unknown vector of parameters when provided as input to the generative component of the model (e.g., the generator of the GAN or the decoder of the VAE) [9]. Thus, the inverse problem is transformed into a minimization problem in the lower-dimensional latent space that can be solved via gradient descent or by adaptively expanding the range of the generator through the optimization of different intermediate layers [10]. This approach, successfully employed in imaging [11] and telecommunications [12], is adapted and leveraged to the TM estimation problem in this paper.

The novelty of this work manifests in two key aspects. Firstly, we contribute by framing traffic matrix estimation as a linear inverse problem and proposing a solution that leverages pre-trained deep generative models as data-driven priors. While this methodology has found success in other domains, it has yet to be adequately investigated in network monitoring and traffic estimation. Specifically, we explore the power of deep generative models to address the ill-posed inverse NT problem of estimating the unobserved traffic matrix from measured link loads. Our approach is data-driven, utilizing historical data comprising previously observed traffic matrices to train the proposed models. We then leverage the trained models to transform the TME task into a more manageable optimization problem in a lower-dimensional latent space. This transformation enables the use of a gradient-based optimizer to efficiently solve the problem and obtain accurate estimates of the traffic matrix in a different manner compared to previous approaches. Secondly, we explore several deep generative model variants in an effort to identify the most suitable neural networks for this data-driven approach and we strive to determine the correct mixture and optimal combination of elements to compose an efficient overall architecture.

The key contributions of this work can be summarized as follows:

- Capitalizing on our previous work [13], we construct a Convolutional Variational Autoencoder (VAE) and leverage it to perform TME by transforming it into a simpler minimization problem in the latent space.
- Aiming to enhance the representation learning ability of the previous model's convolutional layers, we explore for the first time the option of incorporating

attention (i.e., highlighting important spatial locations/correlations) using two alternative architectural units: the Convolutional Block Attention Module (CBAM) [14] or the Squeeze and Excitation (SE) block [15].

- Considering the traffic matrices of a particular network as sequential data (i.e., multivariate time series), we model for the first time the complex spatiotemporal dependencies by employing ConvLSTM [16] layers in the VAE, which combine convolutional operations with the memory capabilities of LSTM.
- Hoping to capture long-range spatiotemporal dependencies across arbitrary distances, we propose employing SA-ConvLSTM [17] layers, which incorporate *self-attention memory* (SAM) into the standard ConvLSTM.
- We implement the proposed deep generative models, exploring all alternative options, and we publish[1] the source code under a permissive free software license, accompanied by comprehensive documentation.
- We extensively evaluate the performance of the proposed models over a comprehensive set of suitable metrics on two publicly available datasets of traffic matrices obtained from real backbone networks. Additionally, we compare our approach with a state-of-the-art method previously published in the literature [18].

The remainder of this paper is organized as follows. In Section II, we briefly overview related work, while in Section III, we introduce the proposed approach and describe in detail the employed deep generative models. Section IV outlines the experimental setup and presents the performance evaluation results and respective discussion. Finally, Section V concludes the paper.

II. RELATED WORK

Traffic matrix estimation has been extensively researched in the last two decades and remains a particularly active topic. Most proposed tomographic methods focus on addressing the ill-posed nature of TME by imposing additional statistical assumptions and leveraging various techniques to reduce the number of free variables. Of course, the apparent consequence of this approach is that the achieved estimation accuracy depends on how well the employed assumptions reflect reality. For example, in [19], all OD flows are assumed independent Poisson random variables. Moreover, given a deterministic routing with a fixed path assigned to each OD flow, the link traffic flows are the superposition of all OD flows that pass through the respective links. Thus, the link traffic flows are also Poisson but not independent. This dependence renders maximum likelihood estimation techniques unsuitable. Instead, the author employs second-order moment matching rate estimation. Extending this initial formulation, a Bayesian rate estimation approach using Markov Chain Monte Carlo simulation is presented in [20], while the authors in [21] replace the Poisson traffic model with a Gaussian traffic model and employ maximum likelihood rate estimation.

Other works incorporate information from additional sources to the linear measurement system. For instance, in [22], the OD flows are assumed proportional to the incoming and outgoing traffic of the nodes, and the so-called *gravity* model is used in combination with SNMP link counts. The devised *tomogravity* method is composed of two stages. First, the TM estimate is initialized using the link counts and the (simple or generalized) gravity model. Then, it is refined using the Moore-Penrose pseudo-inverse to minimize $\|\boldsymbol{Ax}-\boldsymbol{y}\|_2$. On the other hand, in [23], the routing matrix's rank is increased by changing the weight of each link and, therefore, the underlying paths followed by the OD flows. To that end, the authors introduce a temporal model called *route change*, in which each OD flow is considered dependent on its past. Finally, in [24], [25], both spatial and temporal information is leveraged. More precisely, the authors in [24] propose a *Principal Components Analysis* (PCA) method that reduces the dimension of the TME problem by taking into account only the most important eigenflows and a *Kalman* method that employs state space models from the dynamic linear systems theory to capture the evolution of the network. Meanwhile, the authors in [25] introduce the *Sparsity Regularized Matrix Factorization* (SRMF) technique that produces sparse, low-rank approximations of the TM, which are then combined via local interpolation.

More recently, the authors in [26] leverage the *Partial Least Squares* (PLS) regression technique to create a predictive model between the TM and the link counts. They also propose a suitable re-calibration strategy after fixed time intervals to maintain adequate performance. In [27], tomogravity is used to provide an initial TM estimate, which is subsequently improved via *Genetic Algorithm* (GA) optimization using an objective function based on (1). Last but not least, Ephraim et al. [28] extend NT by assuming that the distribution of the counts of each OD flow is a (continuous or discrete) mixture of Poisson distributions, and they estimate the mean traffic rate for every OD pair by solving a second-order moment matching system using least squares and the minimum I-divergence iterative procedure. They also develop a *Moment Generating Function* (MGF) matching approach, which is further evolved to *Characteristic Function* matching [29].

Alternative methods adopt artificial neural networks (ANNs) instead of statistical modeling techniques and additional structural assumptions. These data-driven approaches fall under the supervised learning paradigm and utilize extensive raw measurement data to capture the spatio-temporal patterns of the inverse system and establish a direct mapping between link counts and the traffic matrix. By posing the traffic matrix estimation as a regression problem and defining an appropriate objective function, an iterative optimization algorithm can identify the optimal

1. https://gitlab.com/gkakkavas/gdl-tme

parameters for the neural network architecture. Then, using the trained network, the traffic matrix can be estimated from the observed link loads. Within this context, various neural network models have been proposed for solving the TME problem, including a state-space recurrent multilayer perceptron (RMLP) [30], a back-propagation neural network (BPNN) combined with the iterative proportional fitting procedure (IPFP) [31] or an auto-regressive (AR) model [32], a non-linear auto-regressive exogenous model (NARX) together with the genetic algorithm (GA) [33], and a deep belief network (DBN) [34]. More recent works have attempted to incorporate routing information and topological network structure into the neural network input, such as the Moore-Penrose inverse of the routing matrix multiplied with link load vector [35] and graph-embedding [36], [37], [38]. The latter has been combined with convolutional neural networks [36], [37] and nonnegative matrix factorization (NMF) [38]. A feedforward back-propagation neural network trained with the Levernberg-Marquardt algorithm has also been proposed [39].

Our work also adopts a data-driven approach based on deep neural networks (DNNs) but pertains to the unsupervised learning paradigm—particularly, generative deep learning. Taking this into consideration, the most relevant approach to our work has been presented in [18] because it employs the same problem formulation and enabler for TM estimation (i.e., linear inverse problem and generative deep learning). It involves constructing a solution to the underdetermined TME linear system by leveraging the range space of a properly trained generative model. However, contrary to our methods, the authors in [18] employ a Generative Adversarial Network (GAN) and not a probabilistic model allowing to capture uncertainty like VAE. In addition, they only utilize fully connected layers, they do not make use of any form of attention mechanism (neither convolutional "spatial" attention nor self-attention), and most importantly, they only consider the spatial dependencies among the OD flows of the traffic matrix, ignoring the temporal domain entirely. Our work addresses these limitations by constructing generative models based on the Variational Autoencoder architecture, incorporating specific components tailored to the aforementioned aspects. A detailed description of our approach can be found in the following section.

Generally speaking, we opt to employ variants of the VAE architecture over GAN to capture the inherent uncertainty in the examined problem. While both VAE and GAN are generative models, they differ fundamentally in their learning and data generation approaches. VAE explicitly learns the likelihood distribution through an appropriately defined loss function, while GAN learns through a "min-max two-player game". Another crucial distinction lies in how these models handle latent input variables. In VAE, the latent input variable is associated with a probability distribution that the model learns throughout the learning process. In contrast, GAN's hidden state distributions are predefined. To summarize, machine learning has two primary types of uncertainty: *aleatoric* and *epistemic* [40]. Aleatoric uncertainty, also known as data uncertainty, stems from the inherent variability in the data, leading to uncertainty in predictions. This type of uncertainty is not a model characteristic but rather an intrinsic property of the data distribution, and as such, it is irreducible. On the other hand, epistemic or knowledge uncertainty arises from insufficient knowledge and can, in principle, be reduced (e.g., uncertain predictions for out-of-distribution samples fall into this category). To that end, Bayesian deep learning methods, such as VAEs, endeavor to estimate epistemic uncertainty by modeling distributions for parameter values.

Finally, we note that the task of inferring the network traffic matrix from link load measurements bears similarities to the demand flow estimation problem in transportation, whose objective is to estimate the number of persons or vehicles traveling between specific origins and destinations (OD transit matrix) from the observed flows in particular routes. Numerous studies have investigated this topic, with recent trends leaning towards the application of machine learning, particularly deep learning approaches. Some representative works include the following. In [41], the authors aim to create a theoretically interpretable deep learning approach for estimating different levels of traffic demand by proposing a multi-layered Hierarchical Flow Network (HFN) that can fuse data from diverse sources. The introduction of a special computational graph serves as a modeling tool to extend the HFN and express mathematical formulations. At the same time, the back propagation (BP) algorithm is adapted to compute the derivatives of the underlying composite functions efficiently. Ou et al. [42] present a learning-assigning-searching framework for estimating dynamic OD flows in real time. Their approach involves employing a convolutional neural network as both the learner and assigner, along with two specifically developed genetic algorithms (GAs) functioning as searchers. Finally, in [43], multi-class dynamic OD demand estimation in large-scale transportation networks is framed into a computational graph with tensor representations of spatiotemporal flows, and a forward-backward algorithm is proposed to solve the devised formulation iteratively. Interested readers are directed to [44] for a comprehensive survey of transit OD matrix inference.

III. GENERATIVE DEEP LEARNING-BASED TRAFFIC MATRIX ESTIMATION

In order to solve the problem of estimating the traffic matrix $\boldsymbol{x}$ based on a given measured vector of link counts $\boldsymbol{y}$ and a known routing matrix $\boldsymbol{A}$, we leverage the capabilities of generative modeling. *Generative models* [45] describe how a dataset is generated from a probabilistic perspective. By sampling from such models, we can generate new (i.e., outside the employed training dataset), distinct but similar examples. The probabilistic nature of these models means they must include a stochastic element that influences the generated examples. In other words, the observed data is considered a finite set of samples from an underlying distribution. Any generative model aims to approximate this data distribution using as (training) input the provided data points and then sample from it to generate new, analogous examples that appear as if they could have been included in the original training set. Contrary to discriminative modeling, which is akin to supervised learning (i.e., learning a function that maps an input to an output using a labeled dataset) and regards categorizing samples, generative modeling is not concerned with labeling observations. Instead, it attempts to estimate the likelihood of any given sample. Even though it is usually applied to an unlabeled dataset (i.e., unsupervised learning), it can also employ a labeled dataset to learn how to generate examples belonging to each class. In recent years, deep learning has driven significant advances in generative modeling due to its capability to infer relevant structures directly from the data without any assumptions [46]. The result of this synergy is the creation of the so-called *deep generative models*, which are neural networks with many hidden layers trained to approximate complicated (i.e., unknown or intractable), high-dimensional probability distributions using (independent and identically distributed) samples [7].

The main idea is to transform TME into a constrained minimization problem and explore the latent space of a suitable deep generative model to find a generated TM that best matches the measured link loads. In other words, we constrain the estimated TM to have properties similar to previously observed TMs (i.e., historical data), which are used to train the employed deep generative model. The deep generative models we examine in this work are variations of the Variational Autoencoder (VAE) [47] architecture and will be detailed in the following sections. Generally speaking, the VAE learns the underlying distribution of the historical data and the spatio-temporal traffic patterns via training. Consequently, the respective decoder network can be leveraged to generate synthetic examples $\boldsymbol{x}$ from random low-dimensional latent vectors $\boldsymbol{z}$ that fit the considered network topology and "mimic" the samples of the training set. Among the latter, we select the one that best conforms to the observed link loads. The trained decoder's synthesis capability can also be utilized to create more extensive artificial datasets of "realistic" traffic matrices, which can be used for training and evaluating other relevant methods that require TMs as input.

More rigorously, we assume that the solution can be generated by the trained decoder (i.e., it belongs to the range of the trained decoder), and we leverage the learned distribution to transform TME into the following minimization problem in the lower-dimensional latent space:

$$\arg\min_{z}\Big[\big\|\boldsymbol{y}-\boldsymbol{A}\cdot d(z)\big\|_2^2 + c\cdot\|z\|_2^2\Big], \tag{2}$$

where $d(\cdot)$ denotes the trained decoder and $c\cdot\|z\|_2^2$ is a regularization term motivated by the fact that VAEs impose a Gaussian prior distribution on z, with scalar c weighing the importance of the prior and the measurement error. The intent is to focus on exploring regions of the latent space that the decoder prefers. Simply put, we constrain the solution within the range of the trained generative model and minimize its distance to the observations $\boldsymbol{y}$ to ensure agreement with the measured link counts. This may implicitly introduce bias into the results, as the model is conditioned on historical patterns and may not fully capture potential variations or shifts in the underlying traffic dynamics. One straightforward way to address this limitation is the periodic retraining of the model. This becomes particularly relevant when significant network topology or behavior changes are detected or when estimation accuracy demonstrates substantial deterioration. By updating the model in response to these changes, we can mitigate potential bias and ensure the predictions' continued relevance and accuracy.

To solve the optimization problem (2), we can use a gradient-based optimizer [48] like SGD or Adam since the decoder is differentiable. To speed up the process, we can choose the initial point z_0 of the optimization by iterating over a number M of random latent vectors and selecting the one with the smallest distance to the measured link counts:

$$z_0 = z_k \text{ s.t. } \big\|\boldsymbol{y}-\boldsymbol{A}\cdot d(z_k)\big\|_2^2 \le \big\|\boldsymbol{y}-\boldsymbol{A}\cdot d(z_i)\big\|_2^2, \\ \text{for } i \in 1,\ldots,M. \tag{3}$$

Once we find the optimal $\boldsymbol{z}^*$ that minimizes the objective (2), we can use the mapping $\hat{\boldsymbol{x}} = d(\boldsymbol{z}^*)$ to obtain the estimated TM.

In summary, the proposed generative deep learning-enabled approach for estimating the TM from observed link counts does not require additional assumptions about OD flows, unlike conventional NT TME methods (see Section II). Instead, the necessary prior knowledge is indirectly learned from the historical data used as a training dataset. This reliance on substantial historical/training data is typical in deep learning. A large and diverse dataset is essential for effective learning and generalization, enhancing the models' proficiency in recognizing patterns and making accurate predictions. The performance of deep learning models, including those employed in our work, is directly influenced by the richness and representativeness of the training data. Efforts can be made to reduce the required training data by incorporating more diverse and representative samples (e.g., in our case, TMs spanning all times and days of the week). However, it is crucial to acknowledge the inherent tradeoff between dataset size and model performance/generalization in deep learning approaches. A more extensive and diverse dataset typically leads to improved model performance and better generalization to new, unseen examples. Nevertheless, this advantage comes with the cost of requiring increased amounts of data and computational resources. The following subsections detail the different deep generative models employed in this work.

A. CONVOLUTIONAL VARIATIONAL AUTOENCODER

A Variational Autoencoder (VAE) [49] is a type of unsupervised learning algorithm for learning a lower-dimensional

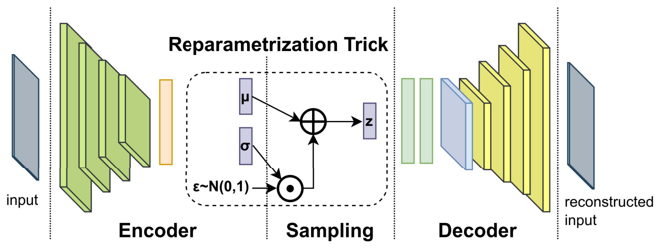


FIGURE 1. Convolutional Variational Autoencoder.

feature representation from unlabeled input data and generating new samples that resemble the training data. Convolutional VAEs, in particular, combine convolutional neural networks (CNNs) and variational autoencoders to efficiently handle data that have a spatial or grid-like structure (e.g., images or, in our case, traffic matrices). As can be seen in Fig. 1, the key components of this generative model are:

- *Encoder*: The encoder network learns the mapping from the input data to a low-dimensional latent space. Contrary to vanilla autoencoders, where each data point $\boldsymbol{x}$ is mapped directly to one point in the latent space (i.e., representation or latent vector), in VAEs, the encoder learns to encode the input data into a distribution in the latent space, $q_\phi(z|\boldsymbol{x})$, typically assumed to be multivariate Gaussian. Then, it produces a latent vector z by sampling from this distribution using the reparameterization trick. The latter allows backpropagation (otherwise, gradients cannot be backpropagated through sampling layers) and efficient end-to-end training. In particular, the encoder outputs this distribution's mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ (in practice, the logarithm of variance that can take any real value, matching the natural output range of a neural network) vectors and then computes the sampled latent vector as
$$z = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \tag{4}$$
where $\boldsymbol{\epsilon}$ is a vector of samples drawn from a standard normal distribution and $\odot$ denotes element-wise multiplication. The encoder network generally consists of several convolutional layers that progressively reduce the dimensionality of the input to capture important features and hierarchical representations. In addition, it often includes BatchNormalization and Dropout layers implementing the respective regularization techniques to improve generalization and prevent overfitting.
- *Decoder*: The decoder learns the mapping back from the latent space to the reconstructed data. It takes the sampled latent vector z as input and aims to reconstruct the original data. In other words, it computes the posterior distribution $p_\theta(\boldsymbol{x}|z)$. The decoder network generally consists of a series of transposed convolutional layers that progressively increase the spatial dimensions of the latent vector to match the original data's dimensions. The final output of the decoder is a reconstructed data point $\hat{\boldsymbol{x}}$ that should closely resemble the original input $\boldsymbol{x}$.
- *Loss Function*: The loss function consists of two components—the reconstruction loss and the regularization loss. The former measures the difference between the original input data and the reconstructed output to make the encoding-decoding scheme efficient. It is typically an element-wise loss, such as the mean squared error (MSE) for continuous real-valued data or the binary cross-entropy for binary data. The latter aims to make the latent space regular by encouraging the learned latent space to "follow" the assumed (Gaussian) distribution. It is computed as the Kullback-Leibler (KL) divergence between the inferred latent distribution $q_\phi(z|\boldsymbol{x})$ and the fixed prior distribution $p(z)$, which is chosen to be $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to evenly distribute the representation vectors around the center of the latent space and to penalize the clustering of points in specific regions. Simply put, the KL divergence term punishes the model for encoding observations into mean and log variance vectors that deviate significantly from the parameters of a standard normal distribution. This type of regularization guarantees that the latent space is both *continuous* (i.e., close points in the latent space produce "similar" content when decoded) and *complete* (i.e., a point sampled from the latent space produces "meaningful" content when decoded).

During training, gradient descent or other related algorithms are used to optimize the parameters of the encoder and decoder networks by minimizing the total loss, which in our case is computed as:
$$\begin{aligned}\mathcal{L} &= \beta \cdot \mathrm{MSE}\big(\boldsymbol{x}, \hat{\boldsymbol{x}}\big) + \mathrm{KL}\Big(q_\phi\big(z|\boldsymbol{x}\big)\|p(z)\Big)\\ &= \beta \cdot \mathrm{MSE}\big(\boldsymbol{x}, \hat{\boldsymbol{x}}\big) + \mathrm{KL}\Big(\mathcal{N}\big(\boldsymbol{\mu}, \boldsymbol{\sigma}\big)\|\mathcal{N}\big(\mathbf{0}, \boldsymbol{I}\big)\Big)\\ &= \beta \cdot \frac{1}{p}\sum_{i=1}^{p}\big(\hat{\boldsymbol{x}}(i) - \boldsymbol{x}(i)\big)^2\\ &\quad + \frac{1}{2}\sum_j\Big(\sigma_j^2 + \mu_j^2 - 1 - \ln\Big(\sigma_j^2\Big)\Big),\end{aligned} \tag{5}$$
where p is the number of OD flows (i.e., the number of elements of $\boldsymbol{x}$), the second summation is over the dimensionality of the latent vector, and the hyperparameter β controls the balance between the reconstruction loss and the regularization term (i.e., KL divergence)—it determines how much weight is given to the reconstruction loss to maintain a balance between the quality of reconstruction and the regularity of the latent space.

B. ATTENTION-ENHANCED CONVOLUTIONAL VAE

Within the context of machine learning, *attention* refers to prioritizing relevant information by selectively attending to salient features of the input while disregarding irrelevant or redundant ones. In this way, the model focuses selectively on the most informative parts and can better handle complex input patterns and capture long-range dependencies. In view of this, we modify the previously described convolutional VAE by integrating a suitable attention mechanism between the convolutional layers hoping to enhance their representation power. Specifically, we explore the use of two alternative architectural units, Convolutional Block Attention Module (CBAM) [14] and Squeeze and Excitation (SE) block [15], which have been shown to achieve state-of-the-art performance.

CBAM aims to improve the flow of information within a convolutional neural network by identifying and highlighting meaningful features in both the channel and spatial dimensions. It is composed of two sequential sub-modules: channel attention and spatial attention. By applying these components in sequence, CBAM enables the network to learn "what" and "where" to attend, improving attention accuracy and noise reduction. Integrating the attention module after every convolutional layer of the overall network allows for the adaptive refinement of the intermediate feature maps. In other words, the features are refined at multiple stages of the network, enhancing the representation power of the model. The *channel attention module* leverages the inter-channel relationships in features to create a channel attention map. Each channel of a feature map $\boldsymbol{F}$ is treated like a feature detector to determine "what" is meaningful for a given input. The module employs max- and average-pooling operations to create two spatial context descriptors. These descriptors are then sent to a shared multi-layer perceptron (MLP) with one hidden layer to calculate the channel attention map $\boldsymbol{M_c}$ through element-wise summation of the two output feature vectors. The *spatial attention module*, on the other hand, exploits the inter-spatial relationships of features to determine "where" the informative parts are located. Average- and max-pooling operations are used along the channel axis, and the results are concatenated to create a single feature descriptor. This descriptor is then forwarded to a standard convolutional layer to produce a 2D spatial attention map $\boldsymbol{M_s}$, highlighting significant spatial locations and suppressing irrelevant ones. Putting everything together, given an intermediate feature map $\boldsymbol{F}$ as input, CBAM sequentially produces an 1D channel attention map $\boldsymbol{M_c}$ and a 2D spatial attention map $\boldsymbol{M_s}$ to calculate the final output as follows:

$$\begin{aligned} \boldsymbol{F'} &= \boldsymbol{M_c}(\boldsymbol{F}) \odot \boldsymbol{F} \text{ and} \\ \boldsymbol{F''} &= \boldsymbol{M_s}(\boldsymbol{F'}) \odot \boldsymbol{F'}, \end{aligned} \tag{6}$$

where $\odot$ denotes element-wise multiplication, $\boldsymbol{F'}$ is the channel-refined feature map, and $\boldsymbol{F''}$ is the final refined output. Fig. 2 illustrates the above process.

Another approach to increase the representational power of the generative model is enhancing spatial encoding by explicitly modeling the interdependencies between the channels of the convolutional features using the Squeeze and Excitation (SE) block. This unit is a lightweight gating mechanism that adaptively recalibrates channel-wise

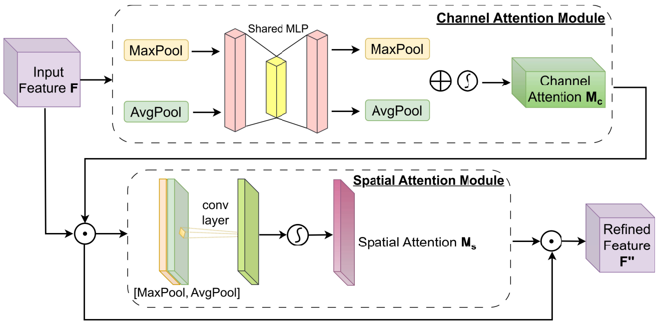


FIGURE 2. Convolutional Block Attention Module (CBAM).

feature responses using global information to selectively emphasize informative features and suppress less useful ones. Specifically, the underlying mechanism comprises two steps: *squeeze* and *excitation*. In the first step, global spatial information is condensed (i.e., squeezed) into a channel descriptor using global average pooling to generate channel-wise statistics. This squeeze operation aggregates the feature maps across spatial dimensions to create a channel descriptor that captures the global distribution of channel-wise feature responses. In this way, information from the global receptive field of the network can be leveraged by its lower layers. Then, the aggregated information is leveraged to fully capture channel-wise dependencies by applying a straightforward gating mechanism with sigmoid activation in the excitation step. This step is realized using two small fully connected (FC) layers and a channel-wise scaling operation. Each channel's excitation is governed by sample-specific activations learned by the self-gating mechanism based on channel dependencies. The feature maps are then appropriately weighted to produce the output of the SE block, which can be directly fed into the following layers.

In summary, we modify the convolutional VAE introduced in Section III-A by integrating attention mechanisms after each convolutional layer. Specifically, we develop two variants: one employing CBAM and another utilizing SE blocks.

C. CONVLSTM-BASED VAE

Due to the nature of network traffic, traffic matrices exhibit spatiotemporal dependencies. In other words, the relationships between spatial locations (OD pairs) evolve over time. Such dependencies arise from various factors:

- The physical layout of a network affects traffic patterns and routing decisions. The traffic between different nodes is interconnected and influenced by the spatial arrangement of the network (i.e., network topology), which can change over time.
- Traffic volumes are affected by the spatial distribution of users, resulting in spatial dependencies in traffic patterns. Meanwhile, temporal dependencies are created due to variations in demand across different periods (e.g., higher demand during peak hours).

- Network conditions and performance can change dynamically, introducing spatiotemporal dependencies in the traffic matrices. For example, congestion or link failures can lead to traffic rerouting and a shift in traffic patterns.

The deep generative models examined so far considered only the spatial dependencies in traffic matrices, which can be effectively captured using convolutional layers thanks to their ability to learn spatial relationships by applying local filters across the input. However, standard convolutional layers cannot inherently handle the temporal dimension. Due to the aforementioned spatiotemporal dependencies, traffic matrices can be viewed as sequential data, with each time step representing a snapshot of the traffic patterns. Thus, it is advantageous to model traffic matrices as multivariate time series and utilize recurrent neural network (RNN) architectures to capture the temporal dimension and effectively model the dependencies between different time steps.

Among the class of RNNs, Long Short-Term Memory (LSTM) has been highly successful in sequence modeling tasks due to its ability to capture long-term dependencies and address the vanishing and exploding gradient problems using its memory cells and gates. Nevertheless, the strengths of CNNs and RNNs should be combined to model spatial and temporal dependencies in data simultaneously. One way to achieve this is through using ConvLSTM [16] layers, a variant of LSTM that replaces the internal matrix multiplications with convolutional operations in the input and recurrent transformations, enabling the processing of spatial information while maintaining sequential modeling capabilities.

In general terms, the flow of information in ConvLSTM is regulated by the input, forget, and output gates, while the cell update gate determines the new cell state. Finally, the updated cell state is combined with the output gate to form the new hidden state. More rigorously, the equations that describe the operations within a ConvLSTM cell in a single time step are the following:

- input gate:

$$i_t = \sigma\big(W_{xi} * x_t + W_{hi} * h_{t-1} + W_{ci} \odot c_{t-1} + b_i\big) \quad (7)$$

- forget gate:

$$f_t = \sigma\big(W_{xf} * x_t + W_{hf} * h_{t-1} + W_{cf} \odot c_{t-1} + b_f\big) \quad (8)$$

- candidate cell state:

$$g_t = \tanh\big(W_{xc} * x_t + W_{hc} \odot h_{t-1} + b_c\big) \quad (9)$$

- cell state:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (10)$$

- output gate:

$$o_t = \sigma\big(W_{xo} * x_t + W_{ho} * h_{t-1} + W_{co} \odot c_t + b_o\big) \quad (11)$$

- hidden state:

$$h_t = o_t \odot \tanh(c_t) \quad (12)$$

where x_t is the input tensor at time step t; h_t is the hidden state tensor at time step t; c_t is the cell state tensor at time step t; $W_{\square i}$, $W_{\square f}$, $W_{\square o}$, and $W_{\square c}$ denote the convolutional kernels for the input gate, forget gate, output gate, and candidate cell state, respectively; $*$ represents the convolution operation; $\odot$ denotes the element-wise multiplication (i.e., Hadamard product); $b_\square$ indicates the bias vectors associated with the respective components; σ is the sigmoid activation function; and tanh is the hyperbolic tangent activation function. ConvLSTM operates sequentially over multiple time steps, updating the hidden state h_t and cell state c_t at each step based on the current input x_t and the previous states.

In conclusion, using ConvLSTM layers in combination with standard convolutional layers in the deep generative model allows us to leverage the strengths of CNNs to extract spatial patterns and the capabilities of LSTMs to capture temporal dependencies and sequential dynamics.

D. SELF-ATTENTION-EXTENDED CONVLSTM-BASED VAE

Self-attention (SA) [50] is a mechanism that allows models to weigh the significance of different elements within a sequence by analyzing their interdependencies. These attention weights determine the contribution of each element to the representation of others, with more relevant parts of the sequence assigned higher importance and less relevant parts lower. The input sequence is processed in parallel, enabling the simultaneous consideration of the dependencies between all elements. By calculating pairwise attention scores and aggregating salient features among all spatial positions, SA can effectively capture long-rage spatiotemporal patterns, attending to different parts of the sequence adaptively.

As discussed earlier, traffic matrices involve complex patterns and dependencies that extend over time and space. An SA mechanism seems promising for effectively capturing these long-range dependencies in traffic matrix estimation. Incorporating self-attention into the deep generative model should make it possible to identify the interactions and correlations between various elements in the sequential data, regardless of their spatial or temporal distance. In order to investigate this, we substitute the ConvLSTM layers in the VAE presented in Section III-C with SA-ConvLSTM [17] layers, which have a Self-Attention Memory (SAM) module incorporated into the regular ConvLSTM.

The Self-Attention Memory (SAM) module utilizes a memory cell to store relevant features from previous time steps to represent global spatiotemporal information. The self-attention mechanism effectively combines current and memorized features by calculating pair-wise similarity scores, determining each element's importance. Additionally, SAM employs a gating mechanism similar to LSTM models to control the flow of information from the memory cell to

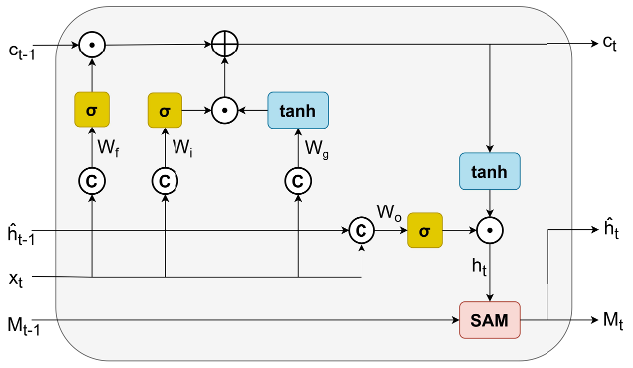


FIGURE 3. Self-Attention ConvLSTM.

the prediction process, allowing for the capture of long-range temporal dependencies by selectively incorporating relevant past features. It receives two inputs: the input feature at the current time step and the memory from the previous time step. The output is an updated representation that enhances the features by considering both the current features and the global spatiotemporal dependencies captured by the memory cell.

The equations that describe the operations within SA-ConvLSTM in a single time step are modified with respect to the ones of standard ConvLSTM as follows:

- ConvLSTM updating:

$$\hat{h}_{t-1} = \mathrm{SAM}(h_{t-1}) \tag{13}$$
$$i_t = \sigma\Big(W_{xi} * x_t + W_{hi} * \hat{h}_{t-1} + W_{ci} \odot c_{t-1} + b_i\Big) \tag{14}$$
$$f_t = \sigma\Big(W_{xf} * x_t + W_{hf} * \hat{h}_{t-1} + W_{cf} \odot c_{t-1} + b_f\Big) \tag{15}$$
$$g_t = \tanh\Big(W_{xc} * x_t + W_{hc} \odot \hat{h}_{t-1} + b_c\Big) \tag{16}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{17}$$
$$o_t = \sigma\Big(W_{xo} * x_t + W_{ho} * \hat{h}_{t-1} + W_{co} \odot c_t + b_o\Big) \tag{18}$$
$$h_t = o_t \odot \tanh(c_t) \tag{19}$$

- SAM module memory updating:

$$i'_t = \sigma\big(W_{m;zi} * Z + W_{m;hi} * h_t + b_{m;i}\big) \tag{20}$$
$$g'_t = \tanh\big(W_{m;zc} * Z + W_{m;hc} \odot h_t + b_{m;c}\big) \tag{21}$$
$$\mathcal{M}_t = \big(1 - i'_t\big) \odot \mathcal{M}_{t-1} + i'_t \odot g'_t \tag{22}$$

- SAM module output:

$$o'_t = \sigma\big(W_{m;zo} * Z + W_{m;ho} * h_t + b_{m;o}\big) \tag{23}$$
$$\hat{h}_t = o'_t \odot \mathcal{M}_t \tag{24}$$

where $W_{m;\square}$ and $b_{m;\square}$ are the weights and bias of the convolutional operators in SAM, $\mathcal{M}_t$ is the memory state of SAM at time t, and Z are the aggregated features computed by the self-attention mechanism. Fig. 3 illustrates the structure of SA-ConvLSTM. If SAM is removed, we regress to the standard ConvLSTM mentioned in Section III-C.

E. IMPLEMENTATION DETAILS

The proposed models were implemented using Python 3.11 and the Keras [51] deep learning API, running on top of the TensorFlow [52] machine learning framework. Specifically, we use the subclassing API to customize models by subclassing the `tf.keras.Model` class. This API provides more control and flexibility over the model's architecture and behavior compared to the sequential and functional API. The constructor specifies the layers and operations comprising each model, while the `call` method defines the model's forward pass. In other words, it defines the model's computation graph by connecting the layers and applying the appropriate operations to the inputs. After compiling and training the models per the standard Keras workflow (i.e., specifying the loss function, optimizer, metrics, and EarlyStopping callback), we use them to make predictions/reconstructions.

Table 1 and Table 2 display the structure of the developed variational autoencoders, including information on the layers and their interconnection, relevant parameters, and output shapes. The latter are reported with respect to the Abilene dataset, whose traffic matrices are of size 12×12. Moreover, the latent space dimensions are set to 8, while for the VAEs incorporating (SA-)ConvLSTM layers, the time series data is prepared via a preprocessing step that divides the continuous sequence into fixed-length, non-overlapping segments or windows. The window length in the displayed output shapes is set to two time steps. The source code of the examined generative models and the complete traffic matrix estimation method (including the approach presented in [18]) is published in https://gitlab.com/gkakkavas/gdl-tme, accompanied by detailed documentation.

IV. PERFORMANCE EVALUATION

A. DATASETS

In order to evaluate the performance of the proposed traffic estimation methods, we conducted numerical experiments over two publicly available datasets comprising actual traffic matrices recorded in real backbone networks.

The first dataset[2] was collected from the Abilene network for a period of 24 weeks. Traffic on Abilene is non-commercial, originating from major universities and research labs throughout the continental United States. The network consists of 12 aggregated nodes located in North America, resulting in 144 traffic pairs (i.e., a 12×12 TM), which were captured in 5-minute intervals every week from 2004-03-01 to 2004-09-10. The network topology comprises 15 undirected edges, each associated with two links. The link capacity is 2 480 000 kbps for the Atlanta-Indianapolis edge and 9 920 000 kbps for all the rest edges. In addition, each node has two external links (one for ingress and one

2. https://www.cs.utexas.edu/∼yzhang/research/AbileneTM/

TABLE 1. Structure of [attention-enhanced] convolutional variational autoencoder*.

ENCODER						
Layer	**Filters/Units**	**Kernel**	**Stride**	**Activation**	**Dropout**	**Output**
Input	-	-	-	-	-	(batch, 12, 12, 1)
Conv2D	32	(3, 3)	(2, 2)	relu	[0.3]	(batch, 6, 6, 32)
[Attention Module]			(CBAM or SE block)			(batch, 6, 6, 32)
Conv2D	64	(3, 3)	(2, 2)	relu	[0.3]	(batch, 3, 3, 64)
[Attention Module]			(CBAM or SE block)			(batch, 3, 3, 64)
Conv2D	128	(3, 3)	(1, 1)	relu	[0.3]	(batch, 3, 3, 128)
[Attention Module]			(CBAM or SE block)			(batch, 3, 3, 128)
Flatten	-	-	-	-	-	(batch, 1152)
Dense	8	-	-	-	-	(batch, 8)
Dense	8	-	-	-	-	(batch, 8)
Sampling	-	-	-	-	-	(batch, 8)
DECODER						
Layer	**Filters/Units**	**Kernel**	**Stride**	**Activation**	**Dropout**	**Output**
Input	-	-	-	-	-	(batch, 8)
Dense	64	-	-	relu	-	(batch, 64)
Dense	576	-	-	relu	-	(batch, 576)
Reshape	-	-	-	-	-	(batch, 3, 3, 64)
Conv2DTranspose	128	(3, 3)	(1, 1)	relu	[0.3]	(batch, 3, 3, 128)
Conv2DTranspose	64	(3, 3)	(2, 2)	relu	[0.3]	(batch, 6, 6, 64)
Conv2DTranspose	32	(3, 3)	(2, 2)	relu	[0.3]	(batch, 12, 12, 32)
Conv2DTranspose	1	(3, 3)	(1, 1)	relu	[0.3]	(batch, 12, 12, 1)

*the components and values inside square brackets refer to the attention-enhanced model

for egress traffic) that connect it to the "outer world", which are not relevant to our analysis. Since the reported traffic data correspond to 5-minute intervals, overall, there are $(\frac{60}{5}) \cdot 24 = 288$ TMs per day and $288 \cdot 7 = 2016$ TMs per week. The provided dataset also includes the routing matrix of the network (i.e., a binary matrix with dimensions 30×144; there are 30 links and 144 OD flows) and the OSPF weight of every link. It does not contain the 30×1 vector of link counts, which can be easily obtained by simple matrix multiplication of the routing matrix with the corresponding traffic matrix (organized in a 144×1 vector).

The second dataset [53] was obtained from GÉANT, a pan-European research network connecting universities and research institutions. It consists of traffic matrices that capture the network traffic between 23 nodes (border routers) interconnected by 38 links. The GÉANT network has 53 additional links to other domains that are not relevant to our analysis. The dataset provides traffic matrices for different days of the week and various hours within a day, constructed using Interior Gateway Protocol (IGP) routing information, sampled Netflow data, and Border Gateway Protocol (BGP) routing information. Specifically, it includes one traffic matrix for every 15-minute interval over a period of approximately four months from 2005-01-01 to 2005-04-23. The traffic matrix values are expressed in kilobits per second (kbps), representing the traffic intensity or volume of data between each pair of nodes.

B. RESULTS AND DISCUSSION

To evaluate the performance of the proposed traffic matrix estimation methods, we measure four key metrics: Root Mean Square Error (RMSE), Normalized Mean Absolute Error (NMAE), Spatial Relative Error (SRE), and Temporal Relative Error (TRE). These metrics quantify the accuracy and relative estimation errors of the OD flows in the traffic matrix and can be computed as follows:

- Root Mean Square Error (RMSE): The RMSE measures the average deviation between the estimated and actual traffic matrix (i.e., the ground truth). It is computed by finding the square root of the mean of the squared differences between corresponding elements of the two matrices. Mathematically, the RMSE can be defined as follows:

$$\mathrm{RMSE}(t) = \frac{\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\|_2}{\sqrt{p}} = \sqrt{\frac{1}{p}\sum_{i=1}^{p}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2},$$

TABLE 2. Structure of [self-attention-extended] ConvLSTM-based variational autoencoder*.

ENCODER						
Layer	Filters/Units	Kernel	Stride	Activation	Dropout	Output
Input	-	-	-	-	-	(batch, 2, 12, 12, 1)
TimeDistributed(Conv2D)	32	(3, 3)	(2, 2)	relu	-	(batch, 2, 6, 6, 32)
TimeDistributed(Conv2D)	64	(3, 3)	(2, 2)	relu	-	(batch, 2, 3, 3, 64)
[SA]ConvLSTM2D	128	(3, 3)	(1, 1)	tanh	0.3	(batch, 2, 3, 3, 128)
Flatten	-	-	-	-	-	(batch, 2304)
Dense	8	-	-	-	-	(batch, 8)
Dense	8	-	-	-	-	(batch, 8)
Sampling	-	-	-	-	-	(batch, 8)
DECODER						
Layer	Filters/Units	Kernel	Stride	Activation	Dropout	Output
Input	-	-	-	-	-	(batch, 8)
Dense	64	-	-	relu	-	(batch, 64)
Dense	1152	-	-	relu	-	(batch, 1152)
Reshape	-	-	-	-	-	(batch, 2, 3, 3, 64)
[SA]ConvLSTM2D	128	(3, 3)	(1, 1)	tanh	0.3	(batch, 2, 3, 3, 128)
TimeDistributed(Conv2DTranspose)	64	(3, 3)	(2, 2)	relu	-	(batch, 2, 6, 6, 64)
TimeDistributed(Conv2DTranspose)	32	(3, 3)	(2, 2)	relu	-	(batch, 2, 12, 12, 32)
TimeDistributed(Conv2DTranspose)	1	(3, 3)	(1, 1)	relu	-	(batch, 2, 12, 12, 1)

*the components inside square brackets refer to the self-attention-extended model

where $\boldsymbol{x}_t$ is the actual traffic matrix at time t, $\hat{\boldsymbol{x}}_t$ is the corresponding estimated traffic matrix, $p \equiv n^2$ is the number of OD flows (i.e., the total number of elements in the matrices), and $i = 1, \ldots, p$ indicates each individual OD flow.

- Normalized Mean Absolute Error (NMAE): The NMAE is a normalized version of the Mean Absolute Error (MAE) and measures the absolute deviation between the estimated and actual traffic matrices at corresponding times. It is calculated by dividing the MAE by the average absolute value of the actual traffic matrix. Mathematically, it is defined as:

$$\mathrm{NMAE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_1}{\left\|\boldsymbol{x}_t\right\|_1} = \frac{\sum_{i=1}^{p} |\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)|}{\sum_{i=1}^{p} |\boldsymbol{x}_t(i)|}.$$

- Spatial Relative Error (SRE): The SRE measures the accuracy of the estimation model at a spatial level. It focuses on each OD flow over its lifetime and determines the relative estimation error. In other words, it quantifies the spatial accuracy of the estimation model. Mathematically, it can be defined as:

$$\mathrm{SRE}(i) = \frac{\left\|\hat{\boldsymbol{x}}_{1:T}(i) - \boldsymbol{x}_{1:T}(i)\right\|_2}{\left\|\boldsymbol{x}_{1:T}(i)\right\|_2} = \frac{\sqrt{\sum_{t=1}^{T}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}}{\sqrt{\sum_{t=1}^{T}\left(\boldsymbol{x}_t(i)\right)^2}},$$

where $\boldsymbol{x}_{1:T}(i)$ is the sequence of the ground truth values of OD flow i across all times, $\hat{\boldsymbol{x}}_{1:T}(i)$ is the sequence of the estimated values of OD flow i across all times, and $t = 1, \ldots, T$ denotes each time.

- Temporal Relative Error (TRE): The TRE evaluates the accuracy of the model's predictions over time by summarizing the relative error between the estimated and actual traffic matrices at each time point. Mathematically, it is defined as:

$$\mathrm{TRE}(t) = \frac{\left\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\right\|_2}{\left\|\boldsymbol{x}_t\right\|_2} = \frac{\sqrt{\sum_{i=1}^{p}\left(\hat{\boldsymbol{x}}_t(i) - \boldsymbol{x}_t(i)\right)^2}}{\sqrt{\sum_{i=1}^{p}\left(\boldsymbol{x}_t(i)\right)^2}}.$$

In the following, the deep generative models are trained using the Adam [54] optimizer on 26 208 TMs collected over thirteen weeks from 2004-05-01 to 2004-07-30 for the Abilene dataset, and 4669 TMs collected over seven weeks from 2005-01-01 to 2005-02-18 for the GÉANT dataset. For testing, 1152 and 582 TMs are used, respectively. Besides serving as the ground truth, these testing TMs are multiplied with the corresponding routing matrices (based on (1) assuming $\boldsymbol{\epsilon}$ is zero) to obtain the testing link load vectors $\boldsymbol{y}$, which are then used during the latent space optimization process as described in (2). In particular, after finding a good initial point $\boldsymbol{z}_0$ in terms of distance to the measured link loads by iterating over $M = 3000$ random

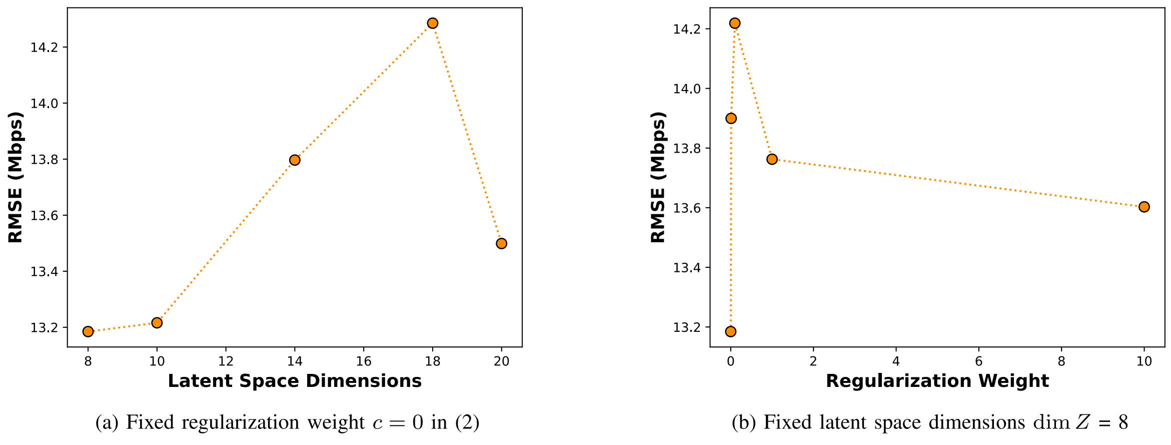


FIGURE 4. Root Mean Square Error across different latent space dimensions and regularization weights on the Abilene dataset.

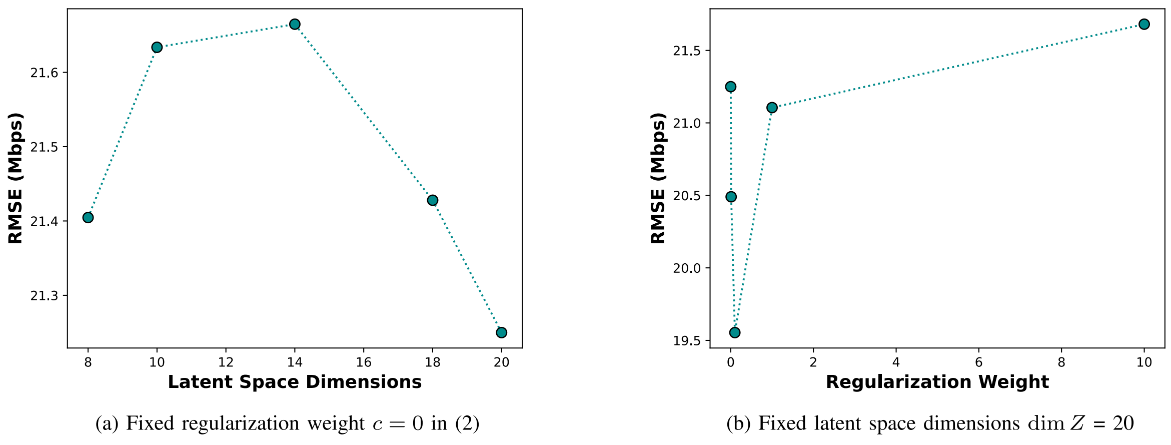


FIGURE 5. Root Mean Square Error across different latent space dimensions and regularization weights on the GÉANT dataset.

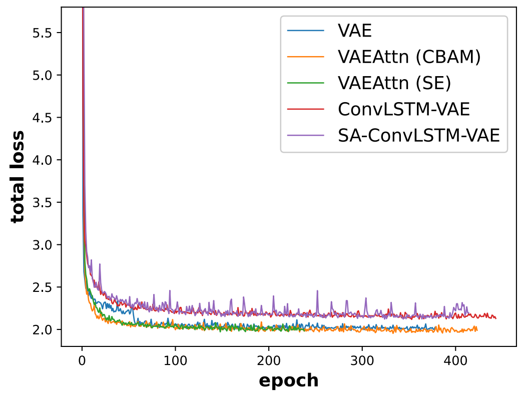
latent vectors according to (2), we perform 5000 iterations for minimization (2) using again the Adam [54] optimization algorithm.

We begin by determining the best values for the latent space dimensions $\dim Z$ (i.e., the number of elements of the latent vector $z \in Z$) and the regularization weight c in (2) for each dataset. To that end, we conduct several small-scale tests on 288 and 192 TMs for the Abilene and the GÉANT datasets, respectively, and compare the achieved RMSEs to select the most appropriate values. Specifically, we first explore the dimensionality of the latent space, considering values of 8, 10, 14, 18, and 20 while keeping the regularization weight fixed at 0. Then, we select the latent dimension resulting in the lowest RMSE and vary the regularization weight among 0, 0.01, 0.1, 1, and 10. Fig. 4 presents the results of this process for the Abilene dataset and Fig. 5 for the GÉANT dataset. As can be seen, the best values are $\{\dim Z = 8, c = 0\}$ for Abilene and $\{\dim Z = 20, c = 0.1\}$ for GÉANT. These values are used in the subsequent experiments involving all the examined deep generative models. Table 3 reports the employed hyperparameters of the proposed generative models for each dataset.

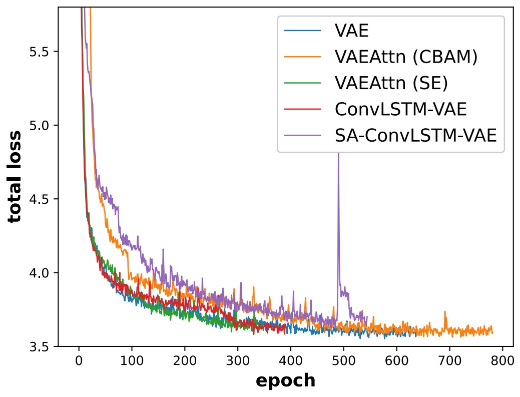
TABLE 3. Hyperparameters of the employed generative models.

Hyperparameter	Abilene	GÉANT
latent space dimensions $\dim Z$	8	20
regularization weight c in (2)	0	0.1
loss scaling factor β in (5)	144 000	5760
early stopping patience	90	60
batch size	128	128
Adam learning rate	0.0005	0.0005
recurrent network window size	2	1

The approach presented in [18], which serves as a benchmark for comparing our proposed models, is built upon a variant of Generative Adversarial Networks known as Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [55]. WGAN-GP is designed to enhance training stability and improve the quality of generated samples by using the Wasserstein-1 distance as the training objective and incorporating a gradient penalty term



(a) Abilene dataset



(b) GÉANT dataset

FIGURE 6. Evolution of total loss along training epochs for the considered generative models on the Abilene and GÉANT datasets.

to regularize the discriminator's gradients. Following the experimental setup described in [18], both the generator and the critic/discriminator of the model are fully connected neural networks with three and four dense layers, respectively. Moreover, the training process lasts 300 epochs, and the discriminator is updated 64 times after each training step of the generator. Lastly, the gradient penalty is set to 10 and the batch size to 64.

Fig. 6 depicts the evolution of total loss, as computed by (5), across the training epochs for the examined generative models. It should be noted that since the early stopping technique is used to prevent overfitting and improve generalization to unseen data, the number of realized training epochs is different for each model. Specifically, the loss metric is monitored during the training process, and if it remains below the minimum value encountered for a specified number of epochs equal to the patience parameter reported in Table 3, training is terminated early.

Table 4 summarizes the results achieved by the proposed generative models and the benchmark method [18] with respect to the four employed evaluation metrics. Figs. 7 and 8 illustrate the temporal relative errors for the TMs of the testing set (each TM corresponds to a time point) and the spatial relative errors of every OD flow, along with the corresponding cumulative distribution functions (CDFs), for the Abilene and the GÉANT dataset, respectively. As can be seen, using the trained decoder of the proposed VAE-based generative models and solving the latent space minimization problem (2) outperforms the GAN-based benchmark method [18] on all four metrics for both datasets.

In the specific case of the Abilene dataset, the integration of attention mechanisms, namely Convolutional Block Attention Module (CBAM) or Squeeze-and-Excitation (SE) block, following each convolutional layer of the VAE, leads to an enhancement in performance across all metrics besides SRE, albeit the magnitude of improvement is modest. As anticipated, the overall best performance is attained by employing ConvLSTM layers in conjunction with standard convolutional layers, effectively capturing spatiotemporal dependencies and sequential dynamics. Finally, even though the introduction of self-attention to the ConvLSTM-based VAE results in a slight deterioration of the RMSE, NMAE, and TRE values when juxtaposed with the vanilla spatiotemporal model, the enhanced model still outperforms the other spatial models and achieves the superior performance in terms of SRE among all considered variants.

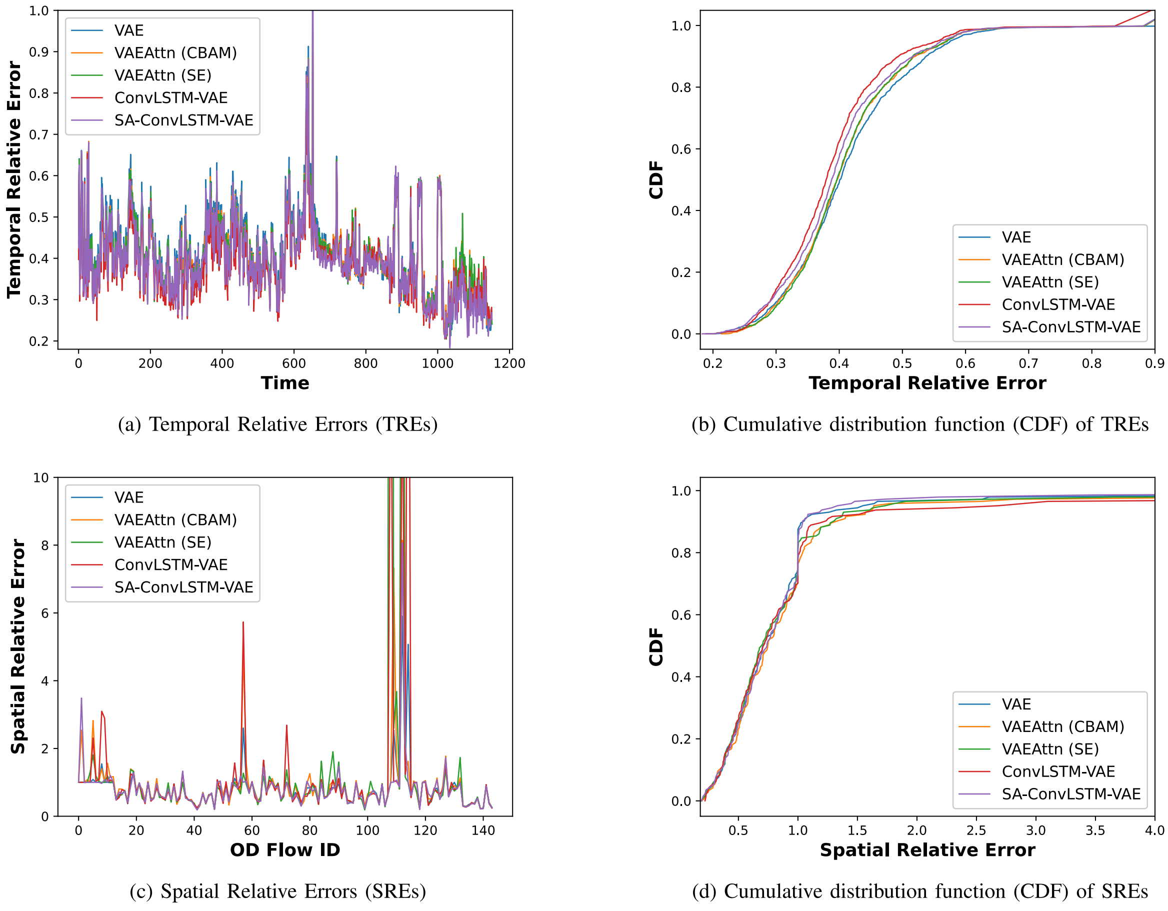
Concerning the GÉANT dataset, the substantial standard deviation and maximum values indicate the presence of outlier data points within each metric, affecting the mean values and complicating the model comparison. However, a more discerning analysis, focusing on median values, conveys a narrative akin to that observed in the Abilene dataset. Specifically, the simple convolutional VAE experiences marginal improvement by including attention mechanisms (CBAM or SE) after each convolutional layer, whereas the ConvLSTM-based VAE achieves the best performance across all metrics, excluding SRE. Once again, introducing self-attention to the ConvLSTM-based VAE significantly improves SRE compared to the vanilla spatiotemporal model. Finally, the larger number of OD flows (576 compared to 144 for Abilene) does explain the observation of more spikes in the spatial relative errors in Fig. 8(c) compared to Fig. 7(c).

We have implemented the benchmark method [18], adhering precisely to the authors' description and details provided in the respective published article. The model's architecture, hyperparameters, and training process were based on the information found in the original work. Consequently, the WGAN-GP model utilizes only feedforward/dense layers. However, recognizing the documented performance benefits of convolutional layers and considering the prevalence of such layers in contemporary GAN architectures, we examine a WGAN-GP variant. Specifically, we modify the generator to include convolutional layers and CBAM or SE blocks. Given that the generator maps from the latent space back to the original domain, it is necessary to double the width

TABLE 4. Estimation errors for the developed deep generative models.

Convolutional Variational Autoencoder (VAE)								
	ABILENE				GÉANT			
Error	Mean	Median	Std	Max	Mean	Median	Std	Max
RMSE (Mbps)	14.79	14.2968	3.7094	37.3193	27.1815	21.4665	25.4697	273.909
NMAE	0.4549	0.4351	0.115	1.3843	0.4969	0.5128	0.1841	2.9225
TRE	0.4113	0.4007	0.098	1.4051	0.456	0.4305	0.1498	2.0266
SRE	0.814	0.7028	0.6659	5.9029	4.7986	1	53.8115	1181.77
Attention-Enhanced Convolutional VAE (CBAM)								
	ABILENE				GÉANT			
Error	Mean	Median	Std	Max	Mean	Median	Std	Max
RMSE (Mbps)	14.6774	14.0338	3.7914	37.3004	26.6066	21.2476	24.8424	273.65
NMAE	0.4718	0.4543	0.1199	1.3966	0.4814	0.4722	0.2594	5.0804
TRE	0.4063	0.3968	0.0913	1.3503	0.4524	0.4221	0.2081	4.1266
SRE	0.9117	0.7455	0.9663	8.1437	5.8503	0.9993	69.6833	1470.52
Attention-Enhanced Convolutional VAE (SE)								
	ABILENE				GÉANT			
Error	Mean	Median	Std	Max	Mean	Median	Std	Max
RMSE (Mbps)	14.7044	13.9428	3.8905	37.0573	26.8623	21.4603	24.9218	284.237
NMAE	0.4493	0.4309	0.1059	1.3417	0.4972	0.4779	0.3603	7.5226
TRE	0.4063	0.3963	0.0905	1.3555	0.4668	0.4251	0.3396	7.5695
SRE	1.5931	0.6848	7.7034	88.7789	1.5506	1	6.2773	122.78
ConvLSTM-Based VAE								
	ABILENE				GÉANT			
Error	Mean	Median	Std	Max	Mean	Median	Std	Max
RMSE (Mbps)	14.0531	13.2033	3.9364	37.086	26.7614	21.0609	25.2564	287.121
NMAE	0.4281	0.4158	0.096	1.2564	0.5014	0.471	0.3991	8.2734
TRE	0.3872	0.3793	0.0866	1.1627	0.4666	0.4198	0.368	8.0807
SRE	1.1395	0.7064	2.4326	22.2894	8.2025	1	134.204	3072.67
Self-Attention-Extended ConvLSTM-Based VAE								
	ABILENE				GÉANT			
Error	Mean	Median	Std	Max	Mean	Median	Std	Max
RMSE (Mbps)	14.3054	13.5421	3.7657	36.2688	27.2948	21.2832	24.979	286.169
NMAE	0.448	0.4324	0.1084	1.3337	0.5274	0.4821	0.4858	10.0405
TRE	0.3967	0.3881	0.0933	1.3196	0.4856	0.4245	0.4802	10.5817
SRE	0.8059	0.7361	0.7205	8.0646	2.8029	1	24.7536	546.867
WGAN-GP [18]								
	ABILENE				GÉANT			
Error	Mean	Median	Std	Max	Mean	Median	Std	Max
RMSE (Mbps)	15.2826	14.4126	3.8476	34.9783	28.6967	23.0073	25.6401	281.313
NMAE	0.485	0.4801	0.0768	1.0287	0.5342	0.5378	0.3336	6.8961
TRE	0.4219	0.4089	0.0863	0.8479	0.4991	0.4659	0.3389	7.5146
SRE	1.1893	0.8153	3.7035	44.9804	1.3284	1	3.4469	62.754

and height of the tensor at (some of the) intermediate layers to reach the original dimensions. In our VAE variants, we use `Conv2DTranspose` layers with a stride of 2 for this purpose, inserting zero values between pixels before performing the convolution operations. In the WGAN-GP generator though, for compatibility with CBAM and SE



(a) Temporal Relative Errors (TREs)

(b) Cumulative distribution function (CDF) of TREs

(c) Spatial Relative Errors (SREs)

(d) Cumulative distribution function (CDF) of SREs

FIGURE 7. Temporal and spatial relative errors for the Abilene dataset.

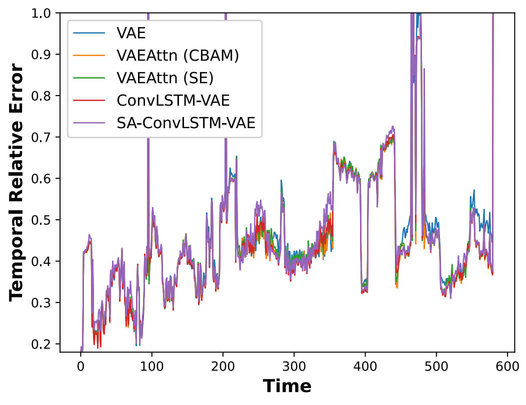
blocks, we employ `UpSampling2D` layers that double the size by repeating each row and column of the input, followed by standard `Conv2D` layers with stride 1 for the convolution operation, and then CBAM or SE blocks for attention. We conducted the same experiments using this variant over the GÉANT dataset and present them in Table 5.

To conclude, in our experimental analysis, we found that the most demanding computational task in the proposed approach is the latent space minimization process, consuming the majority of execution time. When comparing the "spatial" generative models (i.e., the Convolutional VAE and its variants featuring CBAM and SE block) with the WGAN-GP benchmark method, we observe no significant differences in execution time or computational cost. This observation aligns with previous findings in the literature, emphasizing the lightweight and versatile nature of CBAM and the SE block. Specifically, CBAM has been proven end-to-end trainable alongside the convolutional layers it extends, introducing only a small overhead in parameters and computation [14]. Similarly, the SE block slightly increases runtime and computational cost, primarily due to the additional parameters in the two fully connected layers of the gating mechanism. However, these parameters constitute a small fraction of the total network capacity [15].

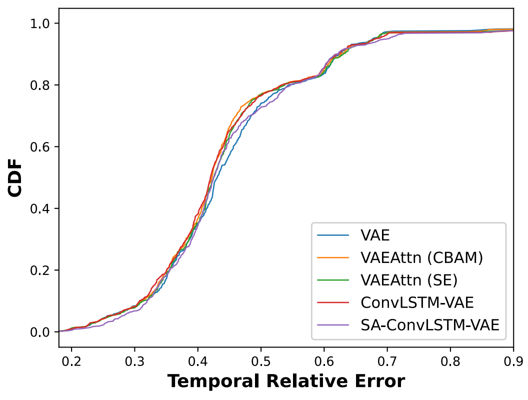
TABLE 5. Estimation errors for the modified WGAN-GP with an attention-enhanced convolutional generator over the GÉANT dataset.

Attention-Enhanced Convolutional WGAN-GP (CBAM)				
Error	**Mean**	**Median**	**Std**	**Max**
RMSE (Mbps)	28.6061	23.5867	25.0741	286.076
NMAE	0.5771	0.5835	0.216	4.1806
TRE	0.4947	0.4682	0.2832	6.1772
SRE	3.6581	1	35.3885	666.718
Attention-Enhanced Convolutional WGAN-GP (SE)				
Error	**Mean**	**Median**	**Std**	**Max**
RMSE (Mbps)	28.3524	23.4138	24.8981	285.885
NMAE	0.5938	0.5918	0.2458	4.9922
TRE	0.488	0.4622	0.2504	5.3451
SRE	6.077	1	66.6591	1355.15

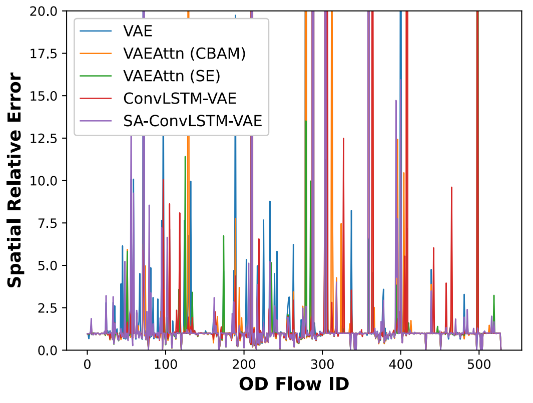
Moreover, it is important to remind that the benchmark method only considers spatial dependencies in traffic matrices. In contrast, the model variant utilizing ConvLSTM layers aims to capture the temporal dimension and model dependencies between different time steps. Consequently,



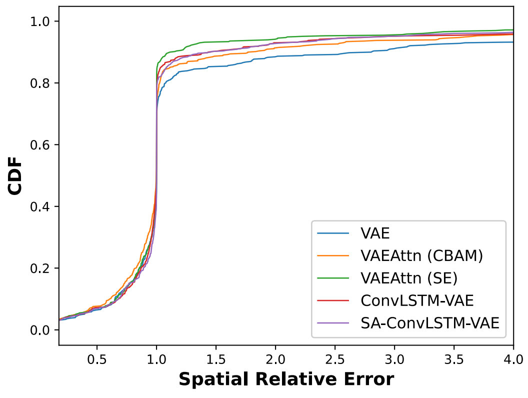
(a) Temporal Relative Errors (TREs)



(b) Cumulative distribution function (CDF) of TREs



(c) Spatial Relative Errors (SREs)



(d) Cumulative distribution function (CDF) of SREs

FIGURE 8. Temporal and spatial relative errors for the GÉANT dataset.

this incurs a higher computational overhead, as is typical in recurrent neural network (RNN)-based architectures. Incorporating self-attention further amplifies the computational burden, requiring correlation calculations across all spatial positions.

V. CONCLUSION

This work explored a novel approach using deep generative models to estimate traffic matrices (TMs) from link load measurements. By leveraging historical TMs, traffic matrix estimation (TME) was simplified and transformed into a constrained minimization problem in a lower-dimensional latent space. The proposed models are variants of the Variational Autoencoder (VAE) architecture and include components such as the Convolutional Block Attention Module (CBAM), the Squeeze and Excitation (SE) block, and ConvLSTM layers with or without self-attention memory for modeling complex spatiotemporal dependencies. They were evaluated on two publicly available datasets of real backbone network traffic matrices and showed improvements in representation learning and accuracy compared to a state-of-the-art method from the literature.

Altogether, the experimental results demonstrated the effectiveness of the deep generative techniques for TME and highlighted the ability of deep generative models to accurately estimate TMs from link load measurements. The presented contributions offer a practical and efficient network monitoring solution, which could enhance network management and traffic engineering. Overall, the proposed deep generative models were able to capture both spatial and temporal dependencies and attend to salient locations within the traffic data.

REFERENCES

[1] P. Tune and M. Roughan, "Internet traffic matrices: A primer," in *Recent Advances in Networking*, vol. 1, H. Haddadi and O. Bonaventure, Eds. New York, NY, USA: ACM SIGCOMM, 2013.

[2] G. Kakkavas, D. Gkatzioura, V. Karyotis, and S. Papavassiliou, "A review of advanced algebraic approaches enabling network tomography for future network infrastructures," *Future Internet*, vol. 12, no. 2, p. 20, Jan. 2020, doi: 10.3390/fi12020020.

[3] G. Kakkavas, A. Stamou, V. Karyotis, and S. Papavassiliou, "Network tomography for efficient monitoring in SDN-enabled 5G networks and beyond: Challenges and opportunities," *IEEE Commun. Mag.*, vol. 59, no. 3, pp. 70–76, Mar. 2021, doi: 10.1109/MCOM.001.2000458.

[4] R. Boutaba et al., "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *J. Internet Serv. Appl.*, vol. 9, no. 1, p. 16, Dec. 2018. [Online]. Available: https://jisajournal.springeropen.com/articles/10.1186/s13174-018-0087-2

[5] T. Panayiotou, M. Michalopoulou, and G. Ellinas, “Survey on machine learning for traffic-driven service provisioning in optical networks,” *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1412–1443, 2nd Quart., 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10050012/

[6] D. Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*, 1st ed. Sebastopol, CA, USA: O’Reilly Media, Inc., 2019, p. on1083570909.

[7] L. Ruthotto and E. Haber, “An introduction to deep generative modeling,” *GAMM-Mitteilungen*, vol. 44, no. 2, Jun. 2021, Art. no. e202100008, doi: 10.1002/gamm.202100008.

[8] A. G. Dimakis, “Deep generative models and inverse problems,” in *Mathematical Aspects of Deep Learning*, 1st ed., P. Grohs and G. Kutyniok, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2022, pp. 400–421. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781009025096

[9] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, “Compressed sensing using generative models,” in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 537–546. [Online]. Available: https://proceedings.mlr.press/v70/bora17a.html

[10] G. Daras, J. Dean, A. Jalal, and A. G. Dimakis, “Intermediate layer optimization for inverse problems using deep generative models,” 2021, *arXiv:2102.07364*.

[11] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett, “Deep learning techniques for inverse problems in imaging,” *IEEE J. Sel. Areas Inf. Theory*, vol. 1, no. 1, pp. 39–56, May 2020, doi: 10.1109/JSAIT.2020.2991563.

[12] E. Balevi, A. Doshi, A. Jalal, A. Dimakis, and J. G. Andrews, “High dimensional channel estimation using deep generative networks,” *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 18–30, Jan. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9252921/

[13] G. Kakkavas, M. Kalntis, V. Karyotis, and S. Papavassiliou, “Future network traffic matrix synthesis and estimation based on deep generative models,” in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Athens, Greece, 2021, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/9522222/

[14] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional block attention module,” in *Proc. 15th Eur. Conf. Comput. Vis.*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham, Switzerland: Springer Int. Publ., 2018, pp. 3–19, doi: 10.1007/978-3-030-01234-2_1.

[15] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 7132–7141.

[16] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, “Convolutional LSTM network: A machine learning approach for precipitation nowcasting,” in *Proc. 28th Adv. Neural Inf. Process. Syst.*, 2015, pp. 802–810. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf

[17] Z. Lin, M. Li, Z. Zheng, Y. Cheng, and C. Yuan, “Self-attention ConvLSTM for spatiotemporal prediction,” in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 11531–11538. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6819

[18] S. Xu, M. Kodialam, T. V. Lakshman, and S. S. Panwar, “Learning based methods for traffic matrix estimation from link measurements,” *IEEE Open J. Commun. Soc.*, vol. 2, pp. 488–499, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9369309/

[19] Y. Vardi, “Network tomography: Estimating source-destination traffic intensities from link data,” *J. Amer. Statist. Assoc.*, vol. 91, no. 433, pp. 365–377, Mar. 1996, doi: 10.1109/TCYB.2021.3062949.

[20] C. Tebaldi and M. West, “Bayesian inference on network traffic using link count data,” *J. Amer. Statist. Assoc.*, vol. 93, no. 442, pp. 557–573, Jun. 1998,” doi: 10.1109/TCYB.2021.3062949.

[21] J. Cao, D. Davis, S. V. Wiel, and B. Yu, “Time-varying network tomography: Router link data,” *J. Amer. Statist. Assoc.*, vol. 95, no. 452, pp. 1063–1075, Dec. 2000, doi: 10.2307/2669743.

[22] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, “Fast accurate computation of large-scale IP traffic matrices from link loads,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 206–217, Jun. 2003, doi: 10.1145/885651.781053.

[23] A. Soule, A. Nucci, R. Cruz, E. Leonardi, and N. Taft, “How to identify and estimate the largest traffic matrix elements in a dynamic environment,” in *Proc. Joint Int. Conf. Meas. Model. Comput. Syst. SIGMETRICS*, 2004, pp. 73–84, doi: 10.1145/1012888.1005698.

[24] A. Soule et al., “Traffic matrices: Balancing measurements, inference and modeling,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 362–373, Jun. 2005, doi: 10.1145/1071690.1064259.

[25] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, “Spatio-temporal compressive sensing and Internet traffic matrices,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 267–278, Aug. 2009, doi: 10.1145/1594977.1592600.

[26] F. J. Cuberos, I. Herrera, K. Wasielewska, and J. Camacho, “Network tomography and partial least squares for traffic matrix estimation,” in *Proc. 17th Int. Conf. Netw. Service Manage. (CNSM)*, 2021, pp. 259–263, doi: 10.23919/CNSM52442.2021.9615551.

[27] J. L. Pachuau, A. Roy, G. Krishna, and A. K. Saha, “Estimation of traffic matrix from links load using genetic algorithm,” *Scalable Comput., Pract. Exp.*, vol. 22, no. 1, pp. 29–38, Feb. 2021, doi: 10.12694/scpe.v22i1.1834.

[28] Y. Ephraim, J. Coblenz, B. L. Mark, and H. Lev-Ari, “Mixed poisson traffic rate network tomography,” in *Proc. 55th Annu. Conf. Inf. Sci. Syst. (CISS)*, 2021, pp. 1–6, doi: 10.1109/CISS50987.2021.9400239.

[29] Y. Ephraim, J. Coblenz, B. L. Mark, and H. Lev-Ari, “Traffic rate network tomography via moment generating function matching,” in *Proc. 56th Annu. Conf. Inf. Sci. Syst. (CISS)*, 2022, pp. 148–153, doi: 10.1109/CISS53076.2022.9751201.

[30] F. Qian, G. Hu, and J. Xie, “A recurrent neural network approach to traffic matrix tracking using partial measurements,” in *Proc. 3rd IEEE Conf. Ind. Electron. Appl.*, 2008, pp. 1640–1643. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4582797

[31] D. Jiang, X. Wang, L. Guo, H. Ni, and Z. Chen, “Accurate estimation of large-scale IP traffic matrix,” *AEU Int. J. Electron. Commun.*, vol. 65, no. 1, pp. 75–86, Jan. 2011, doi: 10.1016/j.aeue.2010.02.008.

[32] D. Jiang, Z. Zhao, Z. Xu, C. Yao, and H. Xu, “How to reconstruct end-to-end traffic based on time-frequency analysis and artificial neural network,” *AEU Int. J. Electron. Commun.*, vol. 68, no. 10, pp. 915–925, Oct. 2014, doi: 10.1016/j.aeue.2014.04.011.

[33] A. Omidvar and H. Shahhoseini, “Intelligent IP traffic matrix estimation by neural network and genetic algorithm,” in *Proc. IEEE 7th Int. Symp. Intell. Signal Process.*, 2011, pp. 1–6, doi: 10.1109/WISP.2011.6051689.

[34] L. Nie, D. Jiang, L. Guo, and S. Yu, “Traffic matrix prediction and estimation based on deep learning in large-scale IP backbone networks,” *J. Netw. Comput. Appl.*, vol. 76, pp. 16–22, Dec. 2016, doi: 10.1016/j.jnca.2016.10.006.

[35] H. Zhou, L. Tan, Q. Zeng, and C. Wu, “Traffic matrix estimation: A neural network approach with extended input and expectation maximization iteration,” *J. Netw. Comput. Appl.*, vol. 60, pp. 220–232, Jan. 2016, doi: 10.1016/j.jnca.2015.11.013.

[36] M. Emami, R. Akbari, R. Javidan, and A. Zamani, “A new approach for traffic matrix estimation in high load computer networks based on graph embedding and convolutional neural network,” *Trans Emerging Tel Tech*, vol. 30, no. 6, Mar. 2019, Art. no. e3604, doi: 10.1002/ett.3604.

[37] R. A. Memon, S. Qazi, and B. M. Khan, “Design and implementation of a robust convolutional neural network-based traffic matrix estimator for cloud networks,” *Wireless Commun. Mobile Comput.*, vol. 2021, pp. 1–11, Jun. 2021. [Online]. Available: https://www.hindawi.com/journals/wcmc/2021/1039613/

[38] S. M. Atif, N. Gillis, S. Qazi, and I. Naseem, “Structured nonnegative matrix factorization for traffic flow estimation of large cloud networks,” *Comput. Netw.*, vol. 201, Dec. 2021, Art. no. 108564. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128621004771

[39] S. S. Hussain, M. A. Sultan, S. Qazi, and M. Ameer, “Intelligent traffic matrix estimation using LevenBerg-marquardt artificial neural network of large scale IP network,” in *Proc. 13th Int. Conf. Math., Actuar. Sci., Comput. Sci. Statist. (MACS)*, 2019, pp. 1–5, doi: 10.1109/MACS48846.2019.9024765.

[40] E. Hüllermeier and W. Waegeman, “Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods,” *Mach. Learn.*, vol. 110, no. 3, pp. 457–506, Mar. 2021. [Online]. Available: https://link.springer.com/10.1007/s10994-021-05946-3

[41] X. Wu, J. Guo, K. Xian, and X. Zhou, "Hierarchical travel demand estimation using multiple data sources: A forward and backward propagation algorithmic framework on a layered computational graph," *Transp. Res. Part C, Emerg. Technol.*, vol. 96, pp. 321–346, Nov. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0968090X18306685

[42] J. Ou, J. Lu, J. Xia, C. An, and Z. Lu, "Learn, assign, and search: Real-time estimation of dynamic origin-destination flows using machine learning algorithms," *IEEE Access*, vol. 7, pp. 26967–26983, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8651455/

[43] W. Ma, X. Pi, and S. Qian, "Estimating multi-class dynamic origin-destination demand through a forward-backward algorithm on computational graphs," *Transportation Research Part C: Emerging Technologies*, vol. 119, Oct. 2020, Art. no. 102747. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0968090X20306604

[44] M. Mohammed and J. Oke, "Origin-destination inference in public transportation systems: A comprehensive review," *Int. J. Transp. Sci. Technol.*, vol. 12, no. 1, pp. 315–328, Mar. 2023. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2046043022000223

[45] H. Gm, M. K. Gourisaria, M. Pandey, and S. S. Rautaray, "A comprehensive survey and analysis of generative models in machine learning," *Comput. Sci. Rev.*, vol. 38, Nov. 2020, Art. no. 100285. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1574013720303853

[46] J. M. Tomczak, *Deep Generative Modeling*. Cham, Switzerland: Springer Int. Publ., 2022. [Online]. Available: https://link.springer.com/10.1007/978-3-030-93158-2

[47] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Found. Trends® Mach. Learn.*, vol. 12, no. 4, pp. 307–392, 2019. [Online]. Available: http://www.nowpublishers.com/article/Details/MAL-056

[48] E. Bisong, "Optimization for machine learning: Gradient descent," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Berkeley, CA, USA: Apress, 2019, pp. 203–207. [Online]. Available: http://link.springer.com/10.1007/978-1-4842-4470-8_16

[49] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013, *arXiv:1312.6114*.

[50] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Adv. Neural Inf. Process. Syst.*, 2017, pp. 1–11. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[51] F. Chollet. "Keras." 2015. [Online]. Available: https://keras.io

[52] M. Abadi et al., "TensorFlow: A sysyem for large-scale machine learning on heterogeneous systems," in *Proc. 12th USENIX Conf. Oper. Syst. Design Implement.*, 2015, pp. 265–283. [Online]. Available: https://www.tensorflow.org/

[53] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, Jan. 2006, doi: 10.1145/1111322.1111341

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[55] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5769–5779.

GRIGORIOS KAKKAVAS (Graduate Student Member, IEEE) received the Diploma degree in electrical and computer engineering from the National Technical University of Athens in 2010, and the M.Sc. degree in management and economics of telecommunications networks from the National and Kapodistrian University of Athens, Greece, in 2012. He is currently pursuing the Ph.D. degree with the NETMODE Laboratory, NTUA, where he is a Research Assistant. His research interests lie in the areas of 5G/6G mobile communications, cognitive radio, and network monitoring techniques, with a focus on network tomography.

NIKOLAOS FRYGANIOTIS received the Diploma degree in electrical and computer engineering from the National Technical University of Athens, Greece, in 2022. He is currently pursuing the Ph.D. degree with the NETMODE Laboratory, NTUA, where he is a Research Assistant. His research interests lie in the areas of distributed randomized graph coloring algorithms, game theory, optimization, machine learning techniques, and reinforcement learning.

VASILEIOS KARYOTIS (Member, IEEE) received the Diploma degree in electrical and computer engineering (ECE) from the National Technical University of Athens (NTUA), Greece, in 2004, the M.Sc. degree in electrical engineering from the University of Pennsylvania, USA, in 2005, and the Ph.D. degree in ECE from NTUA, in 2009. He is currently an Associate Professor with the Department of Informatics, Ionian University, Corfu, Greece. Since 2009, he has been a Research Associate with the NETMODE Laboratory, NTUA, Greece, and since October 2017, he has been an Adjunct Lecturer with Hellenic Open University, Greece. He has coauthored the books *Evolutionary Dynamics of Complex Communications Networks* and *Malware Diffusion Models for Modern Complex Networks: Theory and Applications*. His research interests focus on the modeling and analysis of complex networks, with emphasis on resource allocation, malware propagation, and modeling/control of information diffusion. He was awarded a Fellowship from the Department of ESE of UPenn from 2004 to 2005, and one of two departmental fellowships for exceptional graduate students from the School of ECE of NTUA from 2007 to 2009. He also received a Best Paper Award in ICT 2016.

SYMEON PAPAVASSILIOU (Senior Member, IEEE) is currently a Professor with the School of Electrical and Computer Engineering, National Technical University of Athens, where he is the Director of Network Management and Optimal Design (NETMODE) Laboratory. From 1995 to 1999, he was a Senior Technical Staff Member with AT&T Laboratories, NJ, USA. In August 1999, he joined the ECE Department, New Jersey Institute of Technology, USA, where he was an Associate Professor until 2004. He has an established record of publications with more than 400 technical journal and conference published papers. His main research interests lie in the areas of modeling, optimization and performance evaluation of complex networks and interdependent systems, mobile edge computing, and Internet of Things. He received the Best Paper Award in IEEE INFOCOM 94, the AT&T Division Recognition and Achievement Award in 1997, the U.S. National Science Foundation Career Award in 2003, the Excellence in Research Grant in Greece in 2012, and the Best Paper Awards in IEEE WCNC 2012, ADHOCNETS 2015, ICT 2016, IEEE/IFIP WMNC 2019 and IEEE Globecom 2022, as well as the 2019 IEEE ComSoc Technical Committee on Communications Systems Integration and Modeling Best Paper Award (for his INFOCOM 2019 paper). He has served on several journal editorial positions and held various leading roles in the organization of international conferences and workshops. He also served on the board of the Greek National Regulatory Authority on Telecommunications and Posts from 2006 to 2009.