Rethinking Traffic Prediction in Mobile Network Digital Twins: A flexible inductive graph-based learning model for data-scarce scenarios

Duc-Thinh Ngo, Ons Aouedi, Kandaraj Piamrat, Thomas Hassan, Philippe Raipin

Rethinking Traffic Prediction in Mobile Network Digital Twins: A flexible inductive graph-based learning model for data-scarce scenarios

Duc-Thinh Ngo[a,b,*], Ons Aouedi[c], Kandaraj Piamrat[b], Thomas Hassan[a], Philippe Raipin[a]

[a] *Orange Innovation, Cesson-Sévigné, F-35510, France*
[b] *IMT Atlantique, Nantes University, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, Nantes, F-44000, France*
[c] *SnT, University of Luxembourg, L-4365, Luxembourg*

Abstract

Network Digital Twins (NDTs) are increasingly relying on data-driven approaches for modeling complex network dynamics. Traffic forecasting is crucial for NDTs to provide timely insights for automated network reconfiguration. Existing spatiotemporal forecasting methods, while effective, often rely on pre-constructed graphs, limiting their flexibility in dynamic network environments. This paper introduces FLEX+, an inductive graph-based learning model designed for traffic prediction in data-scarce scenarios. FLEX+ focuses on individual eNodeB traffic prediction by extracting local spatial correlations from k-hop subgraphs, combined with temporal information. Its inductive design allows it to operate on unseen nodes during training, enabling adaptability to evolving network topologies. Empirical studies on a large-scale cellular traffic dataset demonstrate that FLEX+ achieves a 5.9% improvement in accuracy in inductive settings and a 22% reduction in error in data-scarce scenarios when trained with only 3 days of traffic data. Notably, a Knowledge Distillation (KD) framework is introduced to reduce model size and accelerate inference time up to 10 times while maintaining prediction accuracy.

Keywords: Network Digital Twins, Traffic Prediction, Inductive Graph Learning, Knowledge Distillation, Transfer Learning

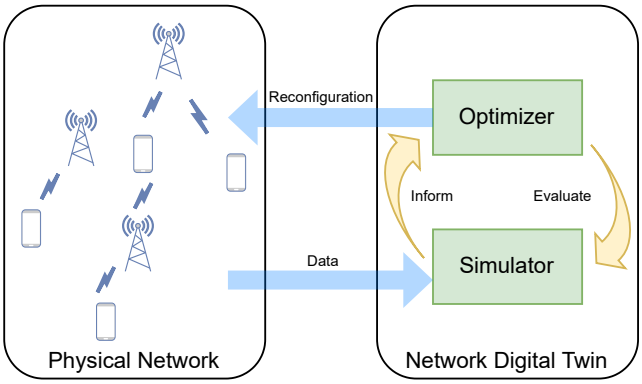


Figure 1: Network Digital Twin (NDT) for automated reconfiguration use case.

1. Introduction

The digital twin is an emerging concept involving building a digital platform to replicate a complex system. Recently, the digital twins of networks, referred to as Network Digital Twins (NDTs), have been widely adopted in research communities as enablers of next-generation networks. Unlike the network simulators, NDTs emphasize the data-driven approach to model complex dynamic networks, facilitated by the two-way data synchronization between the substrate network and its digital replica. Among the various proposed use cases, one significant scenario frequently mentioned is automated network reconfiguration [1–3]. In this scenario, the NDT consists of a data-driven simulator and an optimizer forming a closed loop (Figure 1). The simulator utilizes data collected from the substrate network systems to model the system and provide insights to the optimizer. The optimizer, in turn, may employ various mathematical methods to recommend new network parameters, which are either fed back to the simulator for validation or applied directly on the substrate network.

A persistent challenge in any digital twin system is maintaining fidelity between the digital and physical counterparts due to discrepancies and asynchronization [4]. Traffic prediction is an approach towards solving this issue by attempting to forecast network states, mitigating the impact of delays and reducing reliance on delayed transmitted data. Indeed, to make timely decisions considering the computing delay of several subcomponents, the data-driven simulator must provide the optimizer with future insights rather than just the newly collected data. Therefore, traffic forecasting is critical to keep the NDT informed with accurate predictions of future network status. For example, in O-RAN, cellular traffic prediction can be leveraged to notify the elastic orchestrator whenever there is a need for scaling Virtual Network Functions (VNFs), thereby provisioning the necessary resources [5].

Indeed, the significance of network traffic prediction has been acknowledged not only in the NDT research works [6, 7] but also in a more general context [8–11]. Framed as a time-series forecasting problem, predicting network traffic involves identifying the best model using historical data for accurate fu-

*Corresponding author
Email address: ducthinh.ngo@orange.com (Duc-Thinh Ngo)

ture predictions. Recently, spatiotemporal forecasting based on Graph Neural Networks (GNNs) has emerged as a promising approach, capturing correlations between traffic patterns of Evolved NodeBs (eNBs). For instance, geographically proximate eNBs can exhibit similar traffic volumes due to comparable population densities. Additionally, depending on users' mobility patterns during network usage, network traffic state can be *propagated* from one eNB to another.

Spatiotemporal forecasting typically requires a pre-constructed graph of correlations, often based on eNBs proximity. Existing works [8, 10] conduct spatiotemporal prediction for an entire city in one go, relying on a single proximity graph for the city's network infrastructure. While advantageous for capturing long-range correlations, this approach becomes *rigid* - heavily dependent on the input graph. Therefore, any addition or removal of eNBs requires reinitialization and retraining, incurring extra computing costs. Indeed, while studying the dataset of eNBs [12], we notice that at least one eNB is installed every month in Paris Metropole (Figure 2). Moreover, the limited data from newly deployed eNBs leads to the scarce data scenario for traffic forecasting model training. Hence, transferring the model to a different city poses challenges due to differing network infrastructures.

To address these limitations, in our previous work, we revisited the cellular traffic prediction problem and successfully proposed a novel inductive graph learning model, FLEXIBLE [13], which excelled at limited-training-data forecasting with a straightforward transfer learning framework. In this extended version, now called FLEX+, we enhance the previous model with a new choice for the GraphPooling layer in order to better capture spatial correlation, leading to a different set of hyper-parameters and improved performance. More importantly, we extend the solution notably with a Knowledge Distillation (KD) framework to reduce the model size in data-scarce scenarios while maintaining precision and thereby accelerating inference time. Finally, we conduct additional empirical studies as well.

Our contribution. First of all, focused on forecasting individual eNBs, FLEX+ extracts local spatial correlation from the k-hop subgraph centered at the target eNB, combining it with temporal information for accurate predictions. Its inductive design allows operation on unseen nodes during training, ensuring adaptability. By reframing the problem to predict individual traffic within its local k-hop graph, FLEX+ homogenizes graph topologies and regularizes the spatiotemporal model, enhancing generalizability. This simplifies transfer learning into a direct scheme without additional steps, in contrast to prior methods like [11, 14–17]. Despite the limitations of exploiting only local information, FLEX+ gains the possibility to perform prediction on unseen eNBs and its straightforward transfer learning mechanism. Indeed, by experimental results, we show that when the data for training is very few, a pre-trained FLEX+ that is finetuned on a new city's traffic data can outperform state-of-the-art models. Finally, inspired by the KD framework, we successfully trained a smaller model with limited data while maintaining comparable precision to the original model.

Table 1: Summary of related works on spatiotemporal forecasting and topics that have been addressed.

Ref.	Spatio-temporal	Cellular traffic	Inductive	Transfer learning	Knowledge distillation
[18]	✓				
[19]	✓				
[20]	✓				
[11]	✓	✓		✓	
[14]	✓			✓	
[15]	✓			✓	
[17]	✓			✓	
[21]	✓	✓			✓
[22]	✓	✓			✓
[13]	✓	✓	✓	✓	
Ours	✓	✓	✓	✓	✓

Article organization. The paper is presented as follows: related works are briefly reviewed in Section 2, Section 3.1 introduces the problem formulation and mathematical notations, Section 3.2 describes the methodology on high-level while more details about the model components are given in Section 3.3, empirical studies are reported in Section 4, and finally Section 6 concludes our work and discusses on the future works.

2. Related works

Several works have been proposed to address traffic prediction, not only for cellular traffic but also for road traffic, which serves as a primary source of inspiration for spatiotemporal forecasting. In this section, we review some of these existing works and summarize their explored techniques in Table 1.

2.1. Spatiotemporal forecasting

Spatiotemporal forecasting is an emerging approach to tackle cellular traffic prediction with high precision [8, 10, 11]. This approach draws inspiration from advancements in road traffic forecasting [18, 19, 23], where the integration of spatial and temporal data has led to significant improvements in prediction accuracy. While extensively researched, existing methods typically address the problem in a transductive setting, where the given graph remains constant during both the training and inference phases. This limitation restricts their applicability in dynamic environments where network topologies can change over time. In contrast, this work proposes a more flexible model designed to operate natively in an inductive setting. This design choice allows the model to be adopted in scenarios where the graphs differ between the training and inference phases, thereby enhancing its adaptability to real-world conditions. By leveraging an inductive approach, the proposed model can generalize to unseen nodes and varying graph structures, making it particularly suitable for dynamic and evolving network environments. This flexibility is crucial for practical applications, where network conditions and topologies are rarely static. The ability to handle

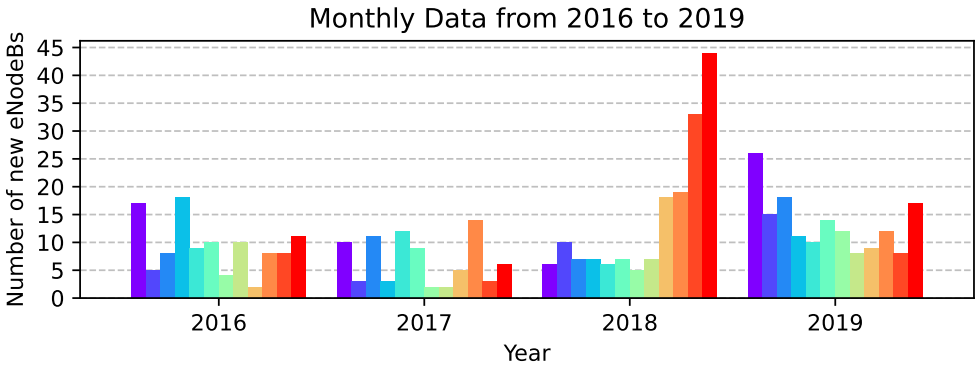


Figure 2: Monthly data visualization of eNB growth. Different colors represent different months.

such variability without compromising prediction accuracy represents a significant advancement over traditional transductive methods.

2.2. Transfer learning

In the domain of spatiotemporal prediction, transfer learning across different cities has been extensively explored, particularly when data is scarce. Researchers have developed various workaround solutions to handle the discrepancies between graph topologies in source and target cities. For instance, some approaches employ graph partitioning with padding [14, 15], which involves dividing the graph into smaller, more manageable subgraphs and padding them to ensure consistency. Other methods utilize node2vec-based feature creation [16], which generates node embeddings that capture the structural properties of the graph, facilitating transfer learning. Additionally, clustering algorithms with inter-city region mapping [11] have been proposed to group similar regions across different cities, thereby enabling the transfer of learned models. Another notable approach is the transferable mechanism for graph structure learning [17], which aims to learn a generalized graph representation that can be applied to different cities. While these methods have shown promise, they introduce additional computing costs and complexity. The most significant drawback, however, is that all these methods necessitate re-training if the target spatial graph changes. This requirement poses a substantial limitation, as it reduces the flexibility and scalability of the models in dynamic environments where network topologies are subject to frequent changes. The need for re-training not only increases computational overhead but also delays the deployment of updated models, making it challenging to maintain high prediction accuracy in real-time applications. Therefore, developing more adaptable and efficient transfer learning techniques remains a critical area of research in spatiotemporal prediction.

2.3. Knowledge distillation

KD is a model compression technique where a smaller student model is trained to replicate the predictions of a larger, more complex teacher model [24]. Instead of learning directly from the ground truth labels, the student model learns from the soft labels generated by the teacher model. These soft labels, which often represent the probability distribution over classes,

Table 2: Table of notation

Notation	Description
$\boldsymbol{X}_i^{(t)}$	Traffic value of i-th node at t-th timestep
dist(i, j)	Geographical distance between locations of i and j
T_h	Number of historical steps
T_f	Number of prediction steps
$\mathcal{T}$	Set of timesteps
$\mathcal{V}$	Set of eNBs
$\boldsymbol{x}_{i,t}$	$\boldsymbol{X}_i^{(t-T_h:t)}$
$\boldsymbol{y}_{i,t}$	$\boldsymbol{X}_i^{(t:t+T_f)}$
$\boldsymbol{h}_i^{(l)}$	Hidden features of node i extracted at the l-th layer
p	Edge dropout probability
d	Dilation factor within the dilated convolution operator
C	Number of hidden features
L	Number of spatiotemporal blocks
λ	Weight decay
$\mathcal{K}$	Set of kernel sizes
B	Batch size
ν	Knowledge distillation coefficient

contain richer information than hard labels, enabling the student model to learn more efficiently and generalize better, particularly in data-scarce scenarios. KD has also been utilized in other contexts; for instance, [21] employs a KD framework to continuously fine-tune the traffic forecasting model with new incoming data, and [22] uses it to enhance collaborative global-local training. While the KD technique has been successful in various applications, it has not been extensively explored for network traffic prediction. In this article, we propose using KD for transfer learning and to reduce the local model size as well as training duration.

3. Proposition

In this section, we describe the problem definition, our proposed methodology, as well as the components of our solution.

3.1. Problem definition

The traffic prediction problem is defined as finding an optimal forecasting function f that predicts the future T_f timesteps given

the historical T_h timesteps. We denote the multivariate time series representing the entire traffic dataset as $\boldsymbol{X}$ and $\boldsymbol{X}_i^{(t)}$ as a scalar value indicating the traffic of the i-th variable at timestep t. The traffic prediction problem is defined as:

$$\arg\min_{f} \sum_{t\in\mathcal{T}, i\in\mathcal{V}} \mathcal{L}\left(f\left(\boldsymbol{X}_i^{(t-T_h:t)}\right), \boldsymbol{X}_i^{(t:t+T_f)}\right), \tag{1}$$

where $\mathcal{L}$ is the loss function, $\mathcal{T}$ is the discrete temporal interval, and $\mathcal{V}$ is the set of vertices - in this case - a set of eNBs. Furthermore, we provide all repeatedly used notations in this paper in the Table 2.

In contrast to previous studies that adopt transductive settings, splitting only $\mathcal{T}$, we introduce Flex+. This fully inductive GNN-based model addresses the continuous deployment of eNBs, allowing variations in both $\mathcal{T}$ and $\mathcal{V}$ during different phases. To achieve this, we reframe the problem as a graph-level task instead of node-level. To predict traffic at base station i during the interval $[t, t+T_f)$, we extract the k-hop subgraph centered at node i. This subgraph, along with its nodes' traffic states, is fed into a learned spatiotemporal model for forecasting, represented by:

$$\hat{\boldsymbol{y}}_{i,t} = f(\boldsymbol{W}_k(i), \{\boldsymbol{x}_{j,t} \mid j \in \mathcal{V}_k(i)\}), \tag{2}$$

where $\boldsymbol{W}_k(i)$ is the adjacency matrix of the induced k-hop subgraph around the node i, $\mathcal{V}_k(i)$ is set of nodes involved in the subgraph, and f is the parameterized model.

3.2. Methodology

To address the forecasting problem raised above, we introduce two learning frameworks to train the Flex+ model. Before delving into the details of each component of the model, this section provides a high-level overview of the two training algorithms: the standard model training and the KD training. The former can be employed in both scenarios - when full data is available and when data is limited. The latter enhances transfer learning, enabling a pretrained Flex+ model to be transferred across cities in data-scarce scenarios.

3.2.1. Standard model training

First of all, we describe the training pipeline of Flex+ as outlined in Algorithm 1. The pipeline is composed of four major stages. Firstly, we extract k-hop subgraphs from the large proximity graph of all eNBs within a city (lines 2-4). This can be considered a data pre-processing stage and we will provide more details in Section 3.3.1. From lines 5 to 23, we enter the training loop, where each iteration can be broken down into three further stages.

At the beginning of each iteration, we sample the mini-batch (from lines 7 to 9), which includes: $\boldsymbol{x}_B$, the set of all historical sequences associated with every node in every k-hop subgraph within the mini-batch; $\boldsymbol{y}_B$, the set of ground truth traffic for targeted nodes; and $\boldsymbol{W}_B$, the stack of all k-hop subgraphs' adjacency matrices within the mini-batch. By diagonally stacking multiple isolated adjacency matrices, we can simplify the numerous computations for each subgraph into a single calculation on a large graph representing the entire mini-batch. Utilizing sparse tensors for implementing adjacency matrices, this mini-batch aggregation introduces no computational or memory overhead and even facilitates model processing.

Algorithm 1 The standard model training pipeline.

Data: $\{\mathcal{G}_i^k = (\mathcal{V}_i^k, \boldsymbol{W}_i^k) \mid i \in \mathcal{V}_{\text{train}}\}, \{(\boldsymbol{x}_{i,t}, \boldsymbol{y}_{i,t}) \mid i \in \mathcal{V}_{\text{train}},\ t \in \mathcal{T}_{\text{train}}\}$

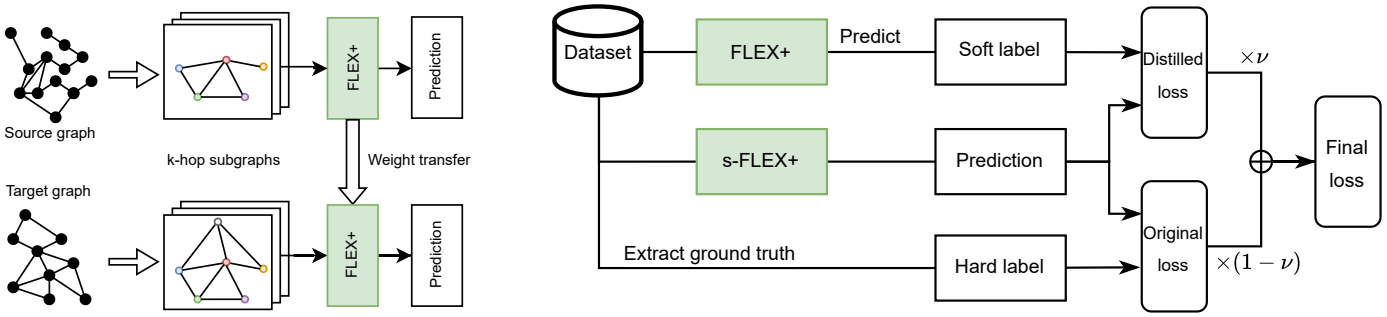
Objective: Obtain optimal $\boldsymbol{\Theta}$ which includes all learnable weights of the model's components as described in Figure 4

1: **procedure** Training
2: **for all** (i,t) in $\mathcal{V}_{\text{train}} \times \mathcal{T}_{\text{train}}$ **do**
3: ▷ *Gathering all traffic sequences of nodes within a k-hop subgraph* ◁
4: $\boldsymbol{x}_{i,t}^{\mathcal{G}} \leftarrow \cup_{j\in\mathcal{V}_i^k}^{\text{stack}} \boldsymbol{x}_{j,t}$
5: **repeat**
6: ▷ *Mini-batch generation* ◁
7: $\mathcal{B} = \{(i,t)\} \leftarrow$ Randomly sample a batch from $\mathcal{V}_{\text{train}} \times \mathcal{T}_{\text{train}}$ with batch size B
8: $\boldsymbol{x}_B, \boldsymbol{y}_B \leftarrow \cup_{(j,u)\in\mathcal{B}}^{\text{concat}} \boldsymbol{x}_{j,u}^{\mathcal{G}}, \cup_{(j,u)\in\mathcal{B}}^{\text{stack}} \boldsymbol{y}_{j,u}$
9: $\boldsymbol{W}_B \leftarrow \text{diag}(\{\boldsymbol{W}_i^k \mid (i,t) \in \mathcal{B}\})$
10: ▷ *Prediction with model* ◁
11: $\boldsymbol{W}_B \leftarrow \text{EdgeDropout}(\boldsymbol{W}_B)$
12: $\boldsymbol{x}_B \leftarrow \text{ReadIn}(\boldsymbol{x}_B)$
13: **for** $l = 1, \ldots, L$ **do**
14: $z_B \leftarrow \text{GraphAgg}_l(\boldsymbol{x}_B, \boldsymbol{W}_B)$
15: $z_B \leftarrow \text{BatchNorm}_l(\text{TCN}_l(z_B))$
16: $\boldsymbol{x}_B \leftarrow \text{ReLU}(z_B + \boldsymbol{x}_B)$
17: $\hat{\boldsymbol{y}}_B \leftarrow \text{GraphPooling}(\boldsymbol{x}_B)$
18: $\hat{\boldsymbol{y}}_B \leftarrow \text{ReadOut}(\hat{\boldsymbol{y}}_B)$
19: ▷ *Gradient descent optimization* ◁
20: $\mathcal{L} \leftarrow \mathcal{L}(\hat{\boldsymbol{y}}_B, \boldsymbol{y}_B)$
21: $\nabla_{\boldsymbol{\Theta}}\mathcal{L} \leftarrow$ Backpropagation
22: Update model weights by gradient descent
23: **until** the early stopping criterion is met

Indeed, when processing the mini-batch, we can consider the input to be the large graph $\boldsymbol{W}_B$ and its node features $\boldsymbol{x}_B$, which are then fed into the spatiotemporal model (lines 11 to 18). The model's output is the prediction of the targeted nodes' traffic, which is subsequently compared to the ground truth using a loss function. This loss function is then utilized to optimize the model weights through backpropagation and gradient descent [25]. In the sections from 3.3.2 to 3.3.7, we will describe some of the model components and the loss function in detail.

3.2.2. Knowledge distillation transfer

Although the transfer learning approach has already proven its advantages when the training data is limited [13], another crucial aspect to consider when building a high-fidelity digital twin of the network is the processing time. In a real-world NDT, traffic prediction provides mathematical optimizers with future insights, enabling the NDT to undergo all necessary processes and recommend new configuration parameters in a timely manner. Therefore, the computing delay of the prediction model is essential for making timely optimization decisions. With this



(a) Transfer learning pipeline

(b) Knowledge distillation (KD) pipeline

Figure 3: The comparison between the transfer learning and the KD framework.

scenario in mind, we propose leveraging the KD framework [24] to distill the vanilla model into a smaller model, thereby accelerating inference without sacrificing a significant amount of prediction precision.

Thanks to the topology-independent nature of FLEX+, the transfer learning is straightforward, as depicted in Figure 3a. With the KD framework, we perform one additional step as outlined in Algorithm 2 and Figure 3b. First, we need to prepare a reduced version of the vanilla model, often considered as the **student** model. In our implementation, we choose to reduce the number of hidden units and the set of convolutional kernels as can be observed in Table 4. The large fine-tuned model, considered as the **teacher**, will be frozen during the training of the student model and generate **soft** labels $\tilde{\boldsymbol{y}}$. These soft labels will be also used to evaluate the model prediction $\hat{\boldsymbol{y}}$ and thereby produce an additional loss term $\mathcal{L}_d = \mathcal{L}(\hat{\boldsymbol{y}}, \tilde{\boldsymbol{y}})$. This additional loss term is combined with the original loss by a weight of ν that we call the **distillation coefficient**:

$$\mathcal{L} = \nu\mathcal{L}(\hat{\boldsymbol{y}}, \tilde{\boldsymbol{y}}) + (1-\nu)\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{3}$$

We can interpret the additional loss term as a way to allow the student model to learn from the richer information contained in the teacher model's soft labels. The higher the distillation coefficient, the softer the final loss, meaning the more significant the role of the teacher model in KD-training the student model.

Algorithm 2 The KD transfer learning pipeline.

Data: $\{\mathcal{G}_i^k = (\mathcal{V}_i^k, \boldsymbol{W}_i^k) | \ i \in \mathcal{V}_{\text{train}}\}, \{(\boldsymbol{x}_{i,t}, \boldsymbol{y}_{i,t}) | \ i \in \mathcal{V}_{\text{train}}, \ t \in \mathcal{T}_{\text{train}}\}$

Teacher model: Large model finetuned on city B $\boldsymbol{f}_{\text{B}}$.

Objective: Train the small model on city B to obtain the optimal $\boldsymbol{f}_B^s$.

procedure KNOWLEDGE DISTILLATION TRANSFER
- $\boldsymbol{f}_B^s \leftarrow$ Random initialization
- **repeat**
  - ▷ *Mini-batch generation* ◁
  - $\boldsymbol{x}_B, \boldsymbol{y}_B, \boldsymbol{W}_B \leftarrow$ Randomly sample a batch.
  - ▷ *Calculate original and distilled loss* ◁
  - $\mathcal{L}_o \leftarrow \mathcal{L}(\boldsymbol{f}_B^s(\boldsymbol{x}_B, \boldsymbol{W}_B), \boldsymbol{y}_B)$
  - $\mathcal{L}_d \leftarrow \mathcal{L}(\boldsymbol{f}_B^s(\boldsymbol{x}_B, \boldsymbol{W}_B), \boldsymbol{f}_B(\boldsymbol{x}_B, \boldsymbol{W}_B))$
  - $\mathcal{L} \leftarrow (1-\nu)\mathcal{L}_o + \nu\mathcal{L}_d$
  - ▷ *Gradient descent optimization* ◁
  - $\nabla_{\boldsymbol{f}_B^s}\mathcal{L} \leftarrow$ Backpropagation
  - Update model weights by gradient descent
- **until** Early stopping criterion is met

3.3. Model components

In this section, we delve deep into important components of the model, which can be found in Figure 4.

3.3.1. Subgraph construction

Proximity graph. Since mobile users are often in movement, we can assume that the nearby eNBs' traffic is correlated. Therefore, we craft a graph of correlation solely based on the geographical position of eNBs:

$$W_{ij}^s = \begin{cases} \exp\left(-\text{dist}(i,j)\right), & \text{if dist}(i,j) < \kappa \text{ and } i \neq j. \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

where dist(i, j) is the geographical distance between two eNBs i and j. In both the Paris and Lyon datasets, we set $\kappa = 3.5$km, which is the minimum distance so that both graphs remain connected. Furthermore, we sparsify the graph by limiting the maximum node degree to 10.

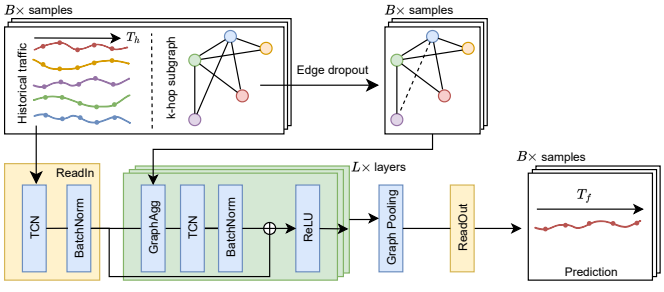


Figure 4: Model architecture which takes a batch of k-hop subgraphs and associated historical traffic to generate a batch of prediction for the target eNB.

k-hop subgraphs. For each node i, we pre-build its k-hop subgraphs from the constructed large proximity graph and store them in a key-value store. Given a pair (i, t), we can efficiently

retrieve the subgraph $\boldsymbol{A}_k(i)$. Additionally, we can access the set of traffic information linked to its vertices at time t using stored node IDs associated with the subgraph. During training, we introduce Edge Dropout to these subgraphs with a probability of p to augment data and mitigate overfitting.

3.3.2. Temporal Convolutional Network

In studying time series data, different methods like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Attention mechanisms are used. Among all of them, the CNN, which is natively designed for parallel computing and requires fewer parameters, is a fast and effective choice for analyzing time series data with good accuracy [18, 19, 23]. For our approach, a key technique is using dilated causal convolution [26]. This helps capture a broad range of information quickly while maintaining the causality in the features we extract. Furthermore, we follow the Inception-like [27] architecture by having multiple kernel sizes in each dilated causal convolution layer to capture features at multiple scales and rapidly enlarge the receptive field. Given $\boldsymbol{h} \in \mathbb{R}^{T_h \times C}$ as the hidden features of an arbitrary node, the proposed Temporal Convolutional Network (TCN) can thus be described as below:

$$\mathrm{TCN}(\boldsymbol{h}) = \bigcup_{K \in \mathcal{K}}^{\text{concat}} \boldsymbol{h} *_{\mathrm{dc}} \boldsymbol{f}_K \tag{5}$$

where $\boldsymbol{f}_K$ is the filter of kernel size K and the resulting feature from TCN is the concatenation of all filtered signals corresponding to every kernel size in the set $\mathcal{K}$. Moreover, $*_{\mathrm{dc}}$ is the dilated causal convolution as described in [26] with the dilation exponentially scaled by a factor d after each layer.

3.3.3. Graph Neural Network

GNN is an important aspect in this study, hence, we consider between two classical yet powerful GNNs for the more efficient overall combination:

- Graph Isomorphism Network (GIN): It has been demonstrated that GNNs can encounter challenges in distinguishing between different graphs in specific cases [28], making them less powerful than the Weisfeiler-Lehman (WL) graph isomorphism test. Addressing this limitation, Xu *et al.* [28] introduced GIN, which has been shown to possess the same discriminative power as the WL test. The model's ability to capture various graph topologies becomes crucial in the context of our forecasting problem formulated as a graph-level task. Distinguishing between different graphs is essential for enhancing the model's expressive power. The graph aggregation in GIN is described as follows:

$$\mathrm{GIN}(\boldsymbol{h}_i) = (1+\epsilon)\boldsymbol{h}_i + \sum_{u \in \mathcal{N}(u)} \boldsymbol{h}_u \tag{6}$$

  where $\boldsymbol{h}_i \in \mathbb{R}^{T_h \times C}$ is the hidden features of an arbitrary node i and ϵ is a learned scalar.

- Graph Attention Network (GAT): Beside GIN, we also benchmark GAT [29] for the GraphAgg components. Unlike GIN, this architecture design prioritizes the possibility of learning edge weights for the aggregation of neighboring nodes, thereby allowing the produce a more expressive graph representation at the cost of complexity:

$$\alpha^{(t)}_{u,v} = \frac{\exp\left(\boldsymbol{a}^{\top}_{(t)}[\boldsymbol{W}\boldsymbol{h}^{(t)}_u \| \boldsymbol{W}\boldsymbol{h}^{(t)}_v]\right)}{\sum_{v' \in \mathcal{N}'(u)} \exp\left(\boldsymbol{a}^{\top}_{(t)}[\boldsymbol{W}\boldsymbol{h}^{(t)}_u \| \boldsymbol{W}\boldsymbol{h}^{(t)}_{v'}]\right)}, \tag{7}$$

$$\mathrm{GAT}(\boldsymbol{h}^{(t)}_i) = \sum_{u \in \mathcal{N}'(i)} \alpha^{(t)}_{i,u} \boldsymbol{h}_u, \tag{8}$$

  where $\boldsymbol{h}^{(t)}_u \in \mathbb{R}^C$ is the hidden features of the node u at time t, $\mathcal{N}'(i) = \mathcal{N}(i) \cup \{i\}$, $\boldsymbol{a}_{(t)} \in \mathbb{R}^{2C}$ and $\boldsymbol{W} \in \mathbb{R}^{C \times C}$ are learned weights, and $\alpha^{(t)}_{u,i}$ can be regarded as the learned weight of the edge between node u and i at time t indicating how much the node u can attend to the node i during the aggregation and vice versa. This equation is inspired by the multi-head GAT [29], considering each timestep as an independent attention head.

However, after conducting the ablation study (Section 4.3.4) to compare these two GNNs, we decided to use GIN for the rest of our experiments. Therefore, whenever Flex+ is mentioned, we refer to the model where GIN serves as the GraphAgg component.

3.3.4. ReadIn

Given the historical traffic $x_{i,t} \in \mathbb{R}^{N \times T_h}$, we process it through a TCN block followed by a batch normalization layer. The extracted features are further processed through a stack of L spatiotemporal layers, as described in Figure 4.

3.3.5. Graph Pooling

The final hidden features are obtained after processing the L-th spatiotemporal layer, resulting in $\boldsymbol{H} \in \mathbb{R}^{N \times T_h \times C}$ corresponding to hidden features of N nodes in the k-hop subgraph. We then apply a graph pooling layer to pool out the final representation that should be used for the prediction. In the previous work, we proposed the TargetPooling for this. While this pooling method allows to take out only the hidden features of the node for prediction, it does not capture the hidden features of neighboring nodes which could add up to the usefulness of the subgraph structure. Therefore, we propose the MixPooling which combines both the Target and Sum pooling with a learnable weight ϵ_p:

$$\mathrm{MixPooling}(\{\boldsymbol{h}_i | i \in \mathcal{V}_k(u)\}) = (1+\epsilon_p)\boldsymbol{h}_i + \epsilon_p \sum_{u \in \mathcal{V}_k(i)} \boldsymbol{h}_u, \tag{9}$$

where i is the index of the target node.

3.3.6. ReadOut

After Graph Pooling, we acquire a hidden feature $\boldsymbol{h} \in \mathbb{R}^{T_h \times C}$. To produce the prediction, we propose a two-step ReadOut module:

$$\hat{\boldsymbol{h}}[\tau, :] = \mathrm{ReLU}\left(\sum_{l=t-T_h}^{t-1} w_{l\tau} \boldsymbol{h}[l, :] + a_\tau\right) \tag{10}$$

$$\hat{\boldsymbol{y}}[\tau] = \sum_{c=0}^{C-1} z_c \hat{\boldsymbol{h}}[\tau, c] + b_\tau \tag{11}$$

where $w_{l\tau}, a_\tau, z_c$, and b_τ are learnable weights.

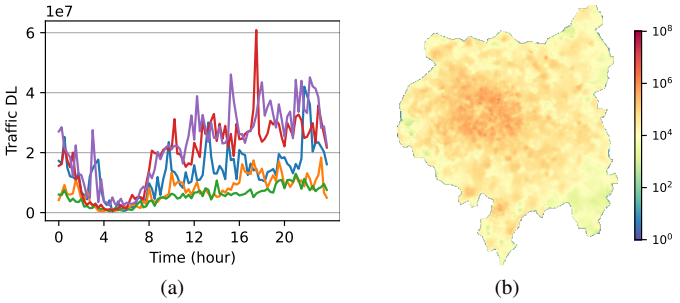


Figure 5: Qualitative examples of the Paris Metropole dataset on March 16th, 2019: (a) Aggregated downlink traffic of 5 random eNBs, (b) Instagram downlink traffic spatially distributed over the Paris Metropole region at 7 a.m.

3.3.7. Loss function

We train the model to minimize the Mean Absolute Arror (MAE) between the prediction and the ground truth while adding a regularization on the norm of the model's parameters to prevent overfitting:

$$\mathcal{L} = \frac{1}{|T_f||\mathcal{T} \times \mathcal{V}|} \sum_{(i,t)\in\mathcal{T}\times\mathcal{V}} \left|\boldsymbol{y}_{i,t} - \hat{\boldsymbol{y}}_{i,t}\right| + \lambda\|\boldsymbol{\Theta}\| \tag{12}$$

where $\boldsymbol{\Theta}$ is the model's learnable parameters and λ is the regularization weight.

4. Performance Evaluation

In this section, we present the dataset used for evaluation, the setup of experiments, and finally the achieved results.

4.1. Dataset

To evaluate the performance of Flex+, we utilize the NetMob dataset [30] which covers 77 days of the traffic demand, from March 16, 2019, to May 31, 2019, generated by 68 mobile services across 20 metropolitan areas in France. This is so far the largest and the most recent cellular traffic dataset that we have found. For our experiments, we aggregate the downlink traffic of 5 prominent services in Paris and Lyon: Apple Video, Fortnite, Netflix, Instagram, and Microsoft Mail. These applications were chosen as they collectively represent the majority of cellular traffic. Moreover, they offer diverse content consumption types, aligned with the Quality of Service Class Identifier [31]. By examining the aggregated traffic of these 5 distinct applications, we can benchmark the model's capability to handle complex traffic dynamics. Figure 5 shows some data samples from the Paris dataset both in temporal and spatial dimensions.

However, the provided data is the traffic per $100 \times 100m^2$ tile which is aggregated from several eNBs' traffic based on distance. This format deviates from the real-world scenario where data is typically collected and aggregated per eNB, which is essential for forecasting traffic and performing radio resource management. To address this issue, we combine the spatiotemporal data with the eNB map [12] and use Voronoi tessellation to re-aggregate the traffic into per-eNB traffic:

$$\text{Vor}(i) = \{\text{tile}_m \mid \text{dist}(\text{tile}_m, i) < \text{dist}(\text{tile}_m, j) \forall j \in \mathcal{V}\} \tag{13}$$

$$\boldsymbol{X}_i^{(t)} = \sum_{m\in\text{Vor}(i)} \boldsymbol{T}_m^{(t)} \tag{14}$$

where $\boldsymbol{T}_m^{(t)}$ is the raw traffic of the m-th tile at time t and $\text{Vor}(i)$ is the Voronoi partition of the i-th eNB. Finally, we obtain 1555 and 1230 eNBs for Paris and Lyon respectively, and 7392 timesteps for each as the data resolution is 1 sample per 15 minutes. We provide more details about the dataset statistics in Table 3.

Table 3: Dataset statistics

Statistics	Paris	Lyon
Number of eNBs	1555	1230
Number of timesteps	7392	7392
max	365.56×10^6	126.56×10^6
min	0	0
mean	14.39×10^6	2.22×10^6
median	10.36×10^6	1.45×10^6

4.2. Experiment settings

4.2.1. Overview

Our model is initially trained on Paris traffic in an inductive setting, serving as a pre-trained model for subsequent transfer to the Lyon dataset. Hyperparameter tuning is carried out during this phase. For fine-tuning Lyon traffic, we adopt a transductive setting to facilitate comparisons with state-of-the-art models. In any setting, data is split into three parts: train, validation, and test sets. Training occurs on the train set, with model selection and hyperparameter tuning based on validation set results. Reported results for comparison come from inference on the test set, following the common setting $T_h = 12, T_f = 3$, which are equivalent to 3 hours and 45 minutes respectively.

4.2.2. Data split

As elaborated in Section 3.1, every data sample can be retrieved via $(i,t) \in \mathcal{T} \times \mathcal{V}$. Accordingly, we split the data by splitting the set $\mathcal{T} \times \mathcal{V}$. We ensure that $|\mathcal{T}_{\text{train}}| = 0.7|\mathcal{T}|$, $|\mathcal{T}_{\text{val}}| = 0.1|\mathcal{T}|$, and $|\mathcal{T}_{\text{test}}| = 0.2|\mathcal{T}|$. The same factors go for $\mathcal{V}$ splitting. Consequently, the 3 data splits in the inductive setting are $\mathcal{T}_{\text{train}} \times \mathcal{V}_{\text{train}}$, $\mathcal{T}_{\text{val}} \times \mathcal{V}_{\text{val}}$, and $\mathcal{T}_{\text{test}} \times \mathcal{V}_{\text{test}}$. In a transductive setting, $\mathcal{V}_{\text{train}} = \mathcal{V}_{\text{val}} = \mathcal{V}_{\text{test}} = \mathcal{V}$.

4.2.3. Evaluation metrics

We adopt two common metrics to evaluate the models' performance. Given the prediction horizon h, the metrics at each horizon are defined as:

- Root Mean Square Error (RMSE):

$$\text{RMSE}_h = \sqrt{\frac{1}{|\mathcal{T}_{\text{test}}||\mathcal{V}_{\text{test}}|} \sum_{i,t} (\boldsymbol{y}_{i,t}[h] - \hat{\boldsymbol{y}}_{i,t}[h])^2} \tag{15}$$

Table 4: Hyperparameters' search space and their optimal values [32].

Hyperparameters	Search space	Large model	Small model
Learning rate	0.001, 0.003, 0.005, ..., 0.019	0.009	0.009
p	0, 0.05, 0.1, 0.15, 0.2	0.1	0.1
d	1, 2, 3	2	2
C	32, 64, 128, 256	256	64
L	1, 2, 3, 4, 5	3	3
λ	$10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}$	10^{-6}	10^{-6}
$\mathcal{K}$	{1, 3}, {3}, {1, 3, 5, 7}	{1, 3, 5, 7}	{1, 3}
B	{256, 512, 1024, 2048, 4096}	2048	2048
Graph Pooling	Mean, Max, Sum, Target, Mix	Mix	Mix
ν	{0.1, 0.5, 0.9}	NA	0.9

- Mean Absolute Arror (MAE):

$$\text{MAE}_h = \frac{1}{|\mathcal{T}_{\text{test}}||\mathcal{V}_{\text{test}}|} \sum_{i,t} \left| \boldsymbol{y}_{i,t}[h] - \hat{\boldsymbol{y}}_{i,t}[h] \right| \qquad (16)$$

4.2.4. Hyperparameter tuning

To effectively tune the model hyperparameters, we use Optuna [32], a hyperparameter search framework. We employ default settings for the optimizer, utilizing the Tree-structured Parzen Estimator algorithm [33] for continual downsampling of the search space. Additionally, the median stopping rule is applied for early pruning of less promising trials. The search space and final hyperparameter values are summarized in Table 4.

4.2.5. Baselines

- Univariate models
  - **LSTM** [34] is a sophisticated neural network architecture meticulously crafted to capture and retain information within long sequential data.
  - **TCN** is a variant of FLEX+, where all graph-related components are removed from the architecture.
- Multivariate models
  - **DCRNN** [19] is a spatiotemporal model, which leverages dual directional diffusion convolution to capture spatial dependencies and incorporates it into a sequence of Gated Recurrent Units to further exploit the temporal dimension.
  - **AGCRN** [23] integrates adaptive graph convolutional networks within a sequence of Gated Recurrent Units (GRUs) to capture both node-specific spatial and temporal correlations in traffic series.
  - **MTGNN** [18] incorporates self-adaptive graph convolution with mix-hop propagation layers for spatial modules and dilated inception layers for temporal modules.
  - **FLEXIBLE** [13] is our previous work, which is the first work to our knowledge to reframe traffic prediction into an inductive graph learning problem.

Table 5: Results of inductive learning on the cellular traffic in Paris and Lyon (the lower the better). The results are averaged over 3 runs for each model.

Cities	Metrics	Horizon	LSTM	TCN	FLEX+
Paris ($\times 10^6$)	MAE	15 min	2.70 ± 0.004	2.71 ± 0.007	**2.62** ± 0.003
		30 min	3.19 ± 0.002	3.21 ± 0.005	**3.04** ± 0.002
		45 min	3.55 ± 0.004	3.56 ± 0.011	**3.34** ± 0.012
	RMSE	15 min	4.53 ± 0.016	4.54 ± 0.019	**4.42** ± 0.021
		30 min	5.26 ± 0.023	5.26 ± 0.010	**5.05** ± 0.021
		45 min	5.64 ± 0.026	5.63 ± 0.008	**5.35** ± 0.005
Lyon ($\times 10^5$)	MAE	15 min	5.17 ± 0.005	5.17 ± 0.016	**5.13** ± 0.007
		30 min	6.06 ± 0.030	6.07 ± 0.021	**5.96** ± 0.017
		45 min	6.68 ± 0.039	6.70 ± 0.019	**6.47** ± 0.002
	RMSE	15 min	9.87 ± 0.134	9.86 ± 0.066	**9.74** ± 0.069
		30 min	11.20 ± 0.076	11.14 ± 0.020	**10.94** ± 0.007
		45 min	12.06 ± 0.054	12.03 ± 0.013	**11.71** ± 0.039

4.3. Results

4.3.1. Inductive learning

This experiment simulates a real-world scenario where a model trained on traffic data of existing eNBs is used to predict traffic on unseen eNBs. Since multivariate baselines are not natively adapted to inductive settings, in this experiment, we only compare our method with univariate models. Conducted on Paris cellular traffic data, the results in Table 5 showcase FLEX+ outperforming other models across all prediction horizons and evaluation metrics. This underscores the advantage of leveraging spatial correlation and FLEX+'s ability to handle graph topology discrepancy between training and inference. Moreover, the comparable performance between LSTM and TCN indicates TCN's effectiveness in capturing sequential information akin to LSTM.

4.3.2. Data-scarcity setting

To assess the forecasting capabilities of newly deployed eNBs, we systematically reduce the volume of training-validation data, investigating the impact on model performance under varying levels of data scarcity: 5%, 10%, 20%, and 40%. In essence, we maintain the test set $\mathcal{T}_{\text{test}}$ unchanged to ensure fair evaluation while progressively dropping the oldest time steps in the training data. By doing so, we ensure that the test data remains unchanged during the entire experiment while guaranteeing that the training data is the most recent which is suitable to the real-world scenario. Moreover, we also keep the training-validation data ratio at 7 over 1. We conduct the benchmark with two types of model training: (1) models that are trained solely on the limited training data of Lyon and (2) models that are trained on the Paris dataset and fine-tuned on the Lyon data (with TR- prefix in the name). The experimental results are finally reported in the Table 6.

Overall, it is observed that the prediction performance degrades as the data becomes more limited, with forecasting error increasing gradually when the amount of training data decreases from 40% to 5% of the full dataset, corresponding to 24 to 3 days of traffic. The three baseline models, MTGNN, DCRNN,

Table 6: MAE of each model in few-shot learning on the cellular traffic in Lyon ($\times 10^5$) (the lower the better).

Scarcity rate	5% (~3 days)			10% (~6 days)			20% (~12 days)			40% (~24 days)		
Horizon (min)	15	30	45	15	30	45	15	30	45	15	30	45
AGCRN	6.95	8.24	9.14	6.26	7.50	8.34	6.06	7.24	8.20	5.25	5.93	6.45
DCRNN	5.72	6.67	7.36	5.41	6.61	7.56	5.37	6.54	7.35	4.92	5.55	5.96
MTGNN	6.38	7.47	8.44	5.80	6.96	8.25	5.62	6.62	7.75	**4.82**	**5.33**	**5.65**
FLEXIBLE	5.91	6.61	7.50	5.34	6.30	7.37	5.28	6.10	6.82	5.39	6.16	6.81
FLEX+	5.59	6.30	6.99	5.72	6.47	7.19	5.24	6.08	6.73	5.25	6.00	6.56
Tr-FLEXIBLE	5.43	6.10	6.64	5.28	6.20	7.03	5.14	5.99	6.67	5.06	5.92	6.49
Tr-FLEX+	**5.15**	**5.96**	**6.54**	**5.12**	**6.05**	**6.77**	**5.07**	**5.89**	**6.57**	5.13	5.93	6.50

and AGCRN, experience a significant increase in prediction error; for instance, MTGNN's error increases from 4.82 to 6.38 on the 15-minute prediction horizon. Meanwhile, FLEXIBLE and FLEX+, which are inductive models, show only a slight degradation in performance, with FLEX+ degrading from 5.25 to 5.59 on the 15-minute horizon. Besides, FLEX+ consistently delivers considerably accurate predictions compared to other baseline models, even when exclusively trained on the Lyon training data. More importantly, our proposed transfer learning framework has demonstrated significant accuracy gains when the data is limited to 12 days or fewer. Among the models in this category, TR-FLEX+ outperforms across all three prediction horizons and on both evaluation metrics. This superior performance highlights the marginal boost achieved through our proposed MixPooling technique and minor tweaks in the training pipeline compared with the previous version.

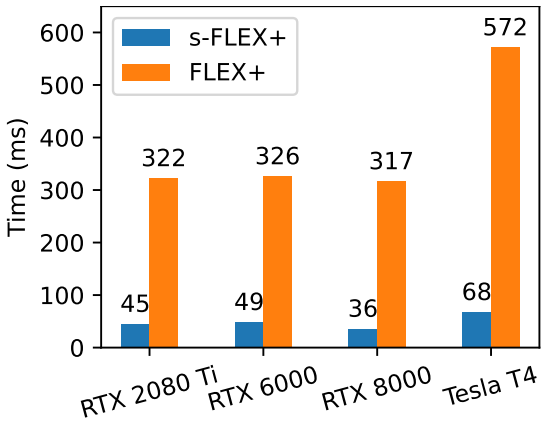


Figure 6: Average inference time of the two models on 1 batch of size 2048 over different hardwares.

4.3.3. Knowledge distillation transfer

In this experiment, we maintained the same data split and ratio of data scarcity as in the previous experiment. Additionally, we compared the standard transferred model with the transferred small model and the KD-based transferred model. The quantitative results are presented in Table 7. We compare the performance among three model variants: Tr-FLEX+ - the large transferred model, Tr-s-FLEX+ - the small model transferred model, and KD-s-FLEX+ - the distilled model. While the first two models follow the straightforward transfer learning framework which is pretrained on Paris and fine-tuned on Lyon, the third one is trained using the KD framework. It is observed that the two small models do not provide predictions as precise as the large model. However, the KD-based model performs significantly better than the model following the vanilla transfer learning scheme and offers comparable performance to the larger model.

In terms of computational time, the smaller model processes predictions approximately 8 times faster than the vanilla model. We benchmarked the processing time of the two models on four different GPUs: RTX 2080 Ti, RTX 6000, RTX 8000, and Tesla T4. The benchmark results are visually displayed in Figure 6. The measured time represents the average inference time for a batch of 2048 eNBs.

In addition, we also conduct other benchmarks to evaluate the training time, CO_2 emissions during training processes and train/inference memory usage of the KD-based finetuning method. We make a comparison to the vanilla fine-tuning model and summarize the results in Table 8. The benchmark for training metrics is measured during the fine-tuning of the models on 20% of training Lyon data. The CO_2-equivalent metrics are evaluated using the codecarbon[1] package. All these experiments are conducted on an RTX 8000 GPU. Since KD requires the vanilla FLEX+ to be pre-fine-tuned with the same training data, its training time and emission are basically the sum of normal FLEX+ training and small model training. Therefore, it is noticed that KD introduces approximately an additional 44 minutes to the total training time and 17g CO_2 to training emissions. To put it in perspective, this 17g of CO_2 is equivalent to the emissions of 2 phones charging[2]. However, KD enables saving GPU memory usage both in training and inference time.

Overall, the KD framework helps to reduce the model size, thereby accelerating inference by up to 8 times while only adding up a 44 minutes of training in the worst case and maintaining a considerable level of prediction precision.

[1] https://github.com/mlco2/codecarbon

[2] https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator

Table 7: MAE of each model in KD learning on the Lyon dataset in the limited data setting (the lower the better).

Scarcity rate	5% (~3 days)			10% (~6 days)			20% (~12 days)			40% (~24 days)		
Horizon (min)	15	30	45	15	30	45	15	30	45	15	30	45
Tr-FLEX+	**5.15**	**5.96**	**6.54**	**5.12**	**6.05**	**6.77**	**5.07**	**5.89**	**6.57**	**5.13**	**5.93**	**6.50**
Tr-s-FLEX+	5.42	6.18	6.76	5.45	6.34	7.08	5.38	6.17	6.85	5.32	6.07	6.65
KD-s-FLEX+	5.29	6.10	6.70	5.19	6.06	6.97	5.08	5.96	6.61	5.18	5.95	6.53

Note: Prefix s- indicates the small model which is also considered as the student model. Unit $\times 10^5$.

Table 8: Comparison of efficiency benchmarks between small and vanilla FLEX+.

Methods	Total train time	Training CO_2eq	Peak GPU train-memory	Peak GPU test-memory
FLEX+	23min	8.37g	4.22 GB	1.74 GB
KD	67min	24.95g	2.72 GB	0.36 GB

4.3.4. Ablation study

In this section, we study the impact of some hyperparameters that we have not tuned using the automated tuner and the architecture choice for the GNN submodule.

Maximum node degree. When constructing the proximity dependency graph from the eNBs' locations, we sparsify the graph by limiting the maximum node degree to 10. To justify this choice, we conducted an ablation study to demonstrate the impact of various maximum node degree choices on the final performance of FLEX+ in inductive settings using the Paris data. The results are visualized in Figure 7.

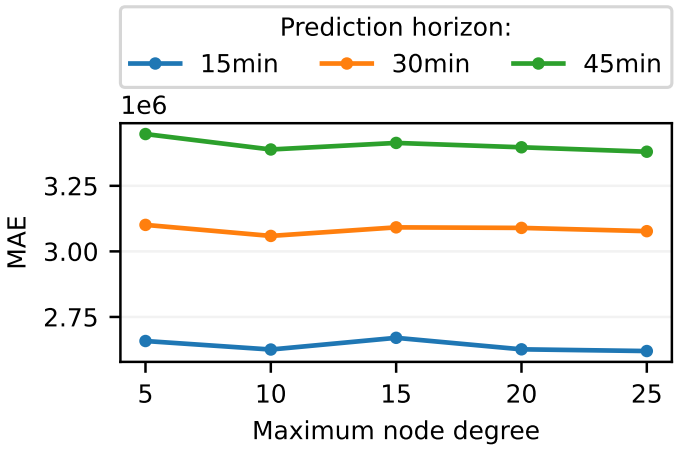


Figure 7: Effect of maximum node degree on the prediction error.

Intuitively, reducing the maximum node degree increases the sparsity of the graph, thereby accelerating computation at the cost of losing some local graph structure. However, a higher limited node degree, which enables a denser graph, does not necessarily improve representation learning due to the potential increase in noise from unrelated neighboring eNBs. Indeed, Figure 7 shows that the maximum node degree has little to no effect on the final performance.

Therefore, we choose to limit the node degree to 10, as it delivers the best performance in the benchmark while maintaining the graph's sparsity to reduce computing costs.

Historical sequence length. In every spatiotemporal forecasting problem, the length of the historical sequence is often set to 12 timesteps, which, in our dataset, equals 3 hours. Intuitively, increasing the historical sequence length can enhance the richness of the information available to the forecaster. However, longer sequences do not always result in better performance, as extending too far back in time may introduce unuseful information or even noise, thereby complicating the prediction and increasing computational complexity. Therefore, one may ask whether simply reducing or lengthening the historical sequence can improve the final performance. Inspired by this idea, we conducted a study to examine the effect of varying the historical sequence length, denoted as T_h, on the final performance. The quantitative results of this study are presented in Figure 8.

In this study, we varied the historical sequence length to 3, 6, 12, 24, and 48 timesteps and presented the results in the inductive settings using the MAE metric across all three prediction horizons. From the plot, it is evident that extending the historical sequence does help to improve prediction performance. The accuracy boost becomes more substantial as the prediction horizon lengthens. However, for the 15-minute horizon, which is equivalent to one-step-ahead forecasting, extending the historical length from 12 to 48 timesteps results in only a marginal performance improvement.

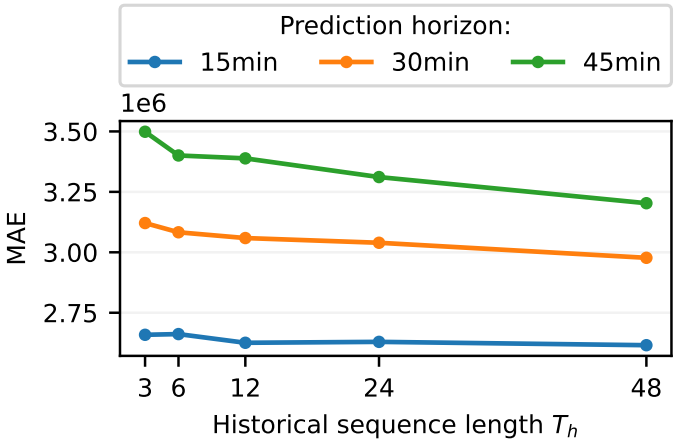


Figure 8: Effect of the historical sequence length on the prediction error.

Table 9: Comparison of prediction error between FLEX+ and TGAT on the Paris dataset in inductive settings. The results are the average of 3 runs. Unit $\times 10^6$.

Metrics		MAE			RMSE			Inference time (ms)	Peak memory allocated (GB)
Horizon		15min	30min	45min	15min	30min	45min		
Model	FLEX+	**2.62** ± 0.003	**3.04** ± 0.002	**3.34** ± 0.012	4.42 ± 0.021	5.05 ± 0.021	**5.35** ± 0.005	**326**	**4.22**
	TGAT	2.63 ± 0.007	3.08 ± 0.011	3.41 ± 0.017	**4.35** ± 0.010	**4.96** ± 0.011	5.40 ± 0.016	421	19.62

Graph Isomorphism vs. Graph Attention Network. We justify our choice of GIN over GAT through an empirical study. Specifically, we replaced the GIN submodule with GAT, as described in Section 3.3.3, to create a new model named TGAT. This model was trained under the same conditions as Flex+, with hyperparameters automatically tuned using Optuna. The results in the inductive setting on the Paris dataset, compared with our Flex+, are presented in Table 9. Additionally, we include a comparison of the empirical time and space complexity of the two models, focusing on inference time and peak memory usage during training. TGAT requires an additional 100ms for inference and consumes up to five times more GPU memory during training. This increase in time and memory complexity comes from the addition of learnable weights $\boldsymbol{W}$ in the self-attention mechanism, whose size is $C \times C$ (refer to the equation 7), while compared to GIN, only one weight ϵ is learned (equation 6). Despite this increased complexity, the TGAT model does not outperform Flex+ which utilizes GIN as its core component.

5. Discussion

5.1. Complexity analysis

In this section, we will attempt to provide an upper-bound theoretical complexity of the model. We first analyze the complexity of each component. To simplify the notation, we denote the input of the model is a graph with V vertices and E edges.

Except for the ReadIn block, the TCN is supposed to produce the same number of features in output as in input; thus, its complexity is $O(T_h K_m C^2)$ where K_m equals the maximum size of all kernels in $\mathcal{K}$.

Since graph convolution is implemented with sparse matrix operations, the GraphAgg operation when using GIN has the complexity of $O(T_h C \sum_{u \in \mathcal{V}} |\mathcal{N}(u)|)$. By the handshaking lemma that states $\sum_{u \in \mathcal{V}} |\mathcal{N}(u)| = 2E$, the complexity of GIN aggregation becomes $O(T_h CE)$. Similarly, if GAT is used in place of GIN, the complexity is notably increased by computing the attention weights and becomes $O(T_h CE + T_h C^2 V)$.

In addition, GraphPooling and ReadOut add up $O(T_h CV)$ and $O(T_h T_f C)$ of complexity, respectively. Other modules such as ReadIn, BatchNorm or ReLU activation do not render notable complexity.

Finally, considering a model of L layers, the upper bound complexity of a forward iteration is $O(LT_h K_m C^2 + LT_h CE + T_h CV + T_h T_f C)$ for Flex+, and: $O(LT_h K_m C^2 + LT_h CE + LT_h C^2 V + T_h T_f C)$ for TGAT. This difference in complexity also explains the increase in the running time of TGAT that we observed in Section 4.3.4.

5.2. Scalability

Using the complexity derived in the previous section, we can discuss the scalability of the model in function of the input graph's characteristics. It is noticed that the complexity linearly depends on the number of vertices and edges of the input graph. During inference, each eNB will require a k-hop subgraph as input to the model. In order to obtain the prediction for all eNBs at once, we process all the k-hop subgraphs of an entire city as a batch. In that case, using mini-batch aggregation as introduced in Section 3.2.1, the input graph is the aggregation of all k-hop subgraphs. Therefore, $V = \sum_{i \in \mathcal{V}} V_i^k$, $E = \sum_{i \in \mathcal{V}} E_i^k$, where V_i^k and E_i^k are, respectively, the number of nodes and edges of the k-hop subgraph surrounding node i.

It is important to note that, while constructing k-hop subgraphs, we limit the maximum degree of the node to a fixed number (section 3.3.1). This helps to lower the number of vertices and edges within V_m and E_m, respectively. Although it is difficult to accurately estimate V and E without any additional information, we can easily bound them by $V \leq |\mathcal{V}|V_m$, $E \leq |\mathcal{V}|E_m$. Therefore, it turns out that the upper bound of inference complexity linearly depends on $|\mathcal{V}|$ which is the number of eNBs.

The aforementioned analyses are performed considering that all eNBs are put together into a batch for a central inference. In reality, we could deploy a distributed inference system where each computing head computes the prediction for a certain number of eNBs. This enables the forecasting system to trade computing resources for time complexity.

5.3. Limitations and future works

In this work, we focus on large urban areas such as Paris and Lyon Metropoles. These regions are often characterized by high-volume traffic, disaggregated areas, and high-mobility users. Although high-volume traffic requires more challenging optimal operations and therefore optimal forecast for NDT systems, disaggregation and frequently mobile users strengthen our assumption of spatial correlation between eNBs. However, in rural areas where network usage does not show these characteristics and use cases, traffic prediction and NDT might find other research challenges that are worth conducting a more thorough investigation. Furthermore, other data about the deployment of eNBs such as height and coverage could be incorporated to further enhance the traffic forecast.

6. Conclusion

This paper presents Flex+, an inductive graph-based learning model for cellular traffic prediction in NDTs. Flex+ effectively

addresses the limitations of existing methods by operating in an inductive setting, enabling accurate predictions on unseen eNBs and facilitating straightforward transfer learning. Empirical studies highlight FLEX+'s superior performance, particularly in data-scarce scenarios. The introduction of a KD framework further enhances FLEX+ by reducing the model size and accelerating inference time without significantly compromising prediction accuracy. Future work will explore incorporating external factors, such as weather and social events, to further improve prediction accuracy. Additionally, investigating the potential of federated learning for collaborative model training across multiple NDTs is a promising direction.

Declaration of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research was performed using data made available by Orange within the NetMob 2023 Data Challenge [30].

References

[1] D.-T. Ngo, O. Aouedi, K. Piamrat, T. Hassan, P. Raipin-Parvédy, Empowering Digital Twin for Future Networks with Graph Neural Networks: Overview, Enabling Technologies, Challenges, and Opportunities, Future Internet 15 (12) (2023) 377. doi:10.3390/fi15120377.

[2] P. Almasan, M. Ferriol-Galmes, J. Paillisse, J. Suarez-Varela, D. Perino, D. Lopez, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, V. Ram, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, P. Barlet-Ros, Network Digital Twin: Context, Enabling Technologies, and Opportunities, IEEE Communications Magazine 60 (11) (2022) 22–27. doi:10.1109/MCOM.001.2200012.

[3] Z.-Q. Luo, X. Zheng, D. López-Pérez, Q. Yan, X. Chen, N. Wang, Q. Shi, T.-H. Chang, A. Garcia-Rodriguez, Srcon: A data-driven network performance simulator for real-world wireless networks, IEEE Communications Magazine 61 (6) (2023) 96–102. doi:10.1109/MCOM.001.2200179.

[4] W. Liu, Y. Fu, Z. Shi, H. Wang, When digital twin meets 6g: Concepts, obstacles, and research prospects, IEEE Communications Magazine 63 (3) (2025) 16–22. doi:10.1109/MCOM.001.2400202.

[5] K. Ali, M. Jammal, Proactive VNF Scaling and Placement in 5G O-RAN Using ML, IEEE Transactions on Network and Service Management 21 (1) (2024) 174–186. doi:10.1109/TNSM.2023.3292986.

[6] H. Nan, R. Li, X. Zhu, J. Ma, D. Niyato, An efficient data-driven traffic prediction framework for network digital twin, IEEE Network 38 (1) (2024) 22–29. doi:10.1109/MNET.2023.3335952.

[7] J. Lai, Z. Chen, J. Zhu, W. Ma, L. Gan, S. Xie, G. Li, Deep Learning Based Traffic Prediction Method for Digital Twin Network, Cognitive Computation 15 (5) (2023) 1748–1766. doi:10.1007/s12559-023-10136-5.

[8] Z. Wang, J. Hu, G. Min, Z. Zhao, Z. Chang, Z. Wang, Spatial-Temporal Cellular Traffic Prediction for 5 G and Beyond: A Graph Neural Networks-Based Approach, IEEE Transactions on Industrial Informatics (2022) 1–10doi:10.1109/TII.2022.3182768.

[9] S. Zhao, X. Jiang, G. Jacobson, R. Jana, W.-L. Hsu, R. Rustamov, M. Talasila, S. A. Aftab, Y. Chen, C. Borcea, Cellular Network Traffic Prediction Incorporating Handover: A Graph Convolutional Approach, in: 2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), IEEE, 2020, pp. 1–9. doi:10.1109/SECON48991.2020.9158437.

[10] Y. Fang, S. Ergut, P. Patras, SDGNet: A Handover-Aware Spatiotemporal Graph Neural Network for Mobile Traffic Forecasting, IEEE Communications Letters 26 (3) (2022) 582–586. doi:10.1109/LCOMM.2022.3141238.

[11] Q. Wu, K. He, X. Chen, S. Yu, J. Zhang, Deep Transfer Learning Across Cities for Mobile Traffic Prediction, IEEE/ACM Transactions on Networking 30 (3) (2022) 1255–1267. doi:10.1109/TNET.2021.3136707.

[12] [Dataset] The National Frequency Agency, Cartoradio: The map of radio sites and wave measurements.
URL https://cartoradio.fr

[13] D.-T. Ngo, K. Piamrat, O. Aouedi, T. Hassan, P. Raipin, Flexible: Forecasting cellular traffic by leveraging explicit inductive graph-based learning, in: 2024 IEEE 35th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), IEEE, 2024, pp. 1–6.

[14] T. Mallick, P. Balaprakash, E. Rask, J. Macfarlane, Transfer learning with graph neural networks for short-term highway traffic forecasting, in: 2020 25th International Conference on Pattern Recognition (ICPR), IEEE, 2021, pp. 10367–10374.

[15] Y. Huang, X. Song, S. Zhang, J. J. Yu, Transfer Learning in Traffic Prediction with Graph Neural Networks, in: 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), IEEE, 2021, pp. 3732–3737. doi:10.1109/ITSC48978.2021.9564890.

[16] Y. Tang, A. Qu, A. H. Chow, W. H. Lam, S. Wong, W. Ma, Domain adversarial spatial-temporal network: A transferable framework for short-term traffic forecasting across cities, in: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, 2022, pp. 1905–1915.

[17] Y. Jin, K. Chen, Q. Yang, Transferable Graph Structure Learning for Graph-based Traffic Forecasting Across Cities, in: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2023, pp. 1032–1043. doi:10.1145/3580305.3599529.

[18] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, C. Zhang, Connecting the dots: Multivariate time series forecasting with graph neural networks, in: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, 2020, pp. 753–763.

[19] Y. Li, R. Yu, C. Shahabi, Y. Liu, Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, in: International Conference on Learning Representations (ICLR '18), 2018.

[20] S. Guo, Y. Lin, N. Feng, C. Song, H. Wan, Attention based spatial-temporal graph convolutional networks for traffic flow forecasting, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 33, 2019, pp. 922–929.

[21] H. Li, J. Wang, C. Hu, X. Chen, X. Liu, S. Jang, G. Dudek, Communication traffic prediction with continual knowledge distillation, in: ICC 2022-IEEE International Conference on Communications, IEEE, 2022, pp. 5481–5486.

[22] K. He, X. Chen, Q. Wu, S. Yu, Z. Zhou, Graph attention spatial-temporal network with collaborative global-local learning for citywide mobile traffic prediction, IEEE Transactions on mobile computing 21 (4) (2020) 1244–1256.

[23] L. Bai, L. Yao, C. Li, X. Wang, C. Wang, Adaptive graph convolutional recurrent network for traffic forecasting, Advances in neural information processing systems 33 (2020) 17804–17815.

[24] G. Hinton, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 (2015).

[25] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, in: International Conference on Learning Representations, 2017.

[26] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, CoRR abs/1609.03499 (2016). arXiv:1609.03499.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9. doi:10.1109/CVPR.2015.7298594.

[28] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: International Conference on Learning Representations, 2019.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph Attention Networks, International Conference on Learning Representations (2018).

URL https://openreview.net/forum?id=rJXMpikCZ
[30] [Dataset] O. E. Martínez-Durive, S. Mishra, C. Ziemlicki, S. Rubrichi, Z. Smoreda, M. Fiore, The netmob23 dataset: A high-resolution multi-region service-level mobile data traffic cartography, CoRR abs/2305.06933 (2023). arXiv:2305.06933, doi:10.48550/ARXIV.2305.06933.
[31] 3GPP TS 23.203 Policy and Charging Control Architecture V17.2.0 (Dec. 2021).
[32] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.
[33] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 24, Curran Associates, Inc., 2011.
[34] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.