Workload Prediction for Volatile Nodes in Multi-Access Edge Networks

Vasilis Avgerinos*, Kostas Ramantas†, Adlen Ksentini‡, Luis Alonso¶, Christos Verikoukis§ *

Email: avgerinos@isi.gr, kramantas@iquadrat.com, adlen.ksentini@eurecom.fr,
luis.alonso@upc.edu, cveri@ceid.upatras.gr

***Abstract*—Advancement of edge and far-edge computing, driven by the increasing demand for real-time, data-intensive applications, has heightened the need for reliable and efficient resource management in volatile environments. This paper introduces GTMixer, a deep learning architecture tailored for predicting resource usage in volatile edge computing scenarios. GTMixer utilizes a Dynamic Temporal Graph (DTG) to capture the evolving interdependencies in workload exchanges across edge and far-edge nodes. By processing snapshots of this graph, GTMixer identifies patterns in resource utilization, even in the absence of historical CPU data. Our contributions include: (1) the creation of a DTG that reflects migration patterns and resource utilization among nodes; (2) the development of the GTMixer model, which integrates feature mixing with graph neural networks for improved predictive accuracy; and (3) empirically evaluating GTMixer against state-of-the-art models using a modified version of the Alibaba 2021 traces dataset that accounts for volatility. Our results demonstrate that GTMixer not only effectively anticipates resource requirements in unpredictable Multi-access Edge Computing (MEC) scenarios but also significantly outperforms current state-of-the-art models in terms of efficiency, showcasing its potential to enhance the reliability and performance of edge computing systems crucial for next-generation network technologies and applications.**

***Index Terms*—6G, Multi-Access Edge Computing, Graph Neural Network, Workload Prediction, Edge/Far-edge,**

I. Introduction

The rapid evolution of digital technologies and the increasing demand for real-time, data-intensive applications have propelled cloud and edge computing into the forefront of modern IT infrastructure. Edge computing emerged as a solution to the latency and bandwidth constraints of traditional cloud environments, bringing computation and data storage closer to the location where it is needed. This transition is crucial for applications requiring immediate processing, such as augmented and virtual reality (AR/VR), telemedicine, intelligent transportation systems, and smart city infrastructures [1]. However, as we transition towards more advanced network Beyond-5G technologies and the conceptualization of 6G, potentially volatile Far Edge nodes (e.g., IoT domains, even UEs) were introduced. This set the stage for a more distributed computing framework, one that must grapple with the inherent volatility and unreliability of edge nodes. In such a landscape, edge nodes—ranging from small-scale IoT devices to larger infrastructure elements—may frequently become unavailable or experience fluctuations in computing capacity, due to factors like energy constraints, maintenance cycles, and dynamic network conditions [2].

As edge computing faces volatile conditions advanced strategies are crucial. These strategies must adeptly anticipate and adapt to the unpredictable nature of edge resources. Moreover, with the advent of 6G networks, which will host both Virtual Network Functions (VNFs) and general microservices atop the infrastructure, the critical role of decoupling traffic and CPU usage becomes apparent. Traffic alone is no longer a reliable indicator of CPU usage, necessitating more sophisticated predictive techniques that include enhanced information exchange between nodes to ensure efficient resource prediction.

To solve the above problems, in this paper, we propose a new deep learning model architecture called GTMixer that operates on temporal graphs to predict resource usage for volatile nodes. First, we construct a Dynamic Temporal Graph (DTG) that highlights the dynamic interdependencies, characteristic of workload and data exchange in edge servers; workload is generated by microservices placed at the edge and cloud servers, including potentially volatile far edge devices (Figure 1). GTMixer then takes as input snapshots of the DTG to effectively capture and analyze the evolving relationships and interactions between nodes. By mapping the exchange of microservices and their workloads across various nodes and utilizing a graph neural network model, the GTMixer can predict the resource usage patterns of physical nodes, even in the absence of historical CPU usage data. The contributions of this paper are summarized as follows:

- Construction of a dynamic relational graph between physical nodes, from a information exchange point of view.
- GTMixer: A novel Graph Neural Network (GNN) architecture that combines feature mixing for time series prediction with the message passing scheme of GNN theory.
- Introduction of a new benchmark constructed from the Alibaba 2021 traces dataset, where volatile nodes, which were missing from SoTA works, are also considered.

The paper is organized into five sections. Firstly, Section II

*Industrial Systems Institute, Patras, Greece, †Iquadrat Informatica, Barcelona, Spain, ‡EURECOM, Sophia Antipolis, France, §University of Patras, Greece, ¶Universitat Politècnica de Catalunya, Barcelona, Spain

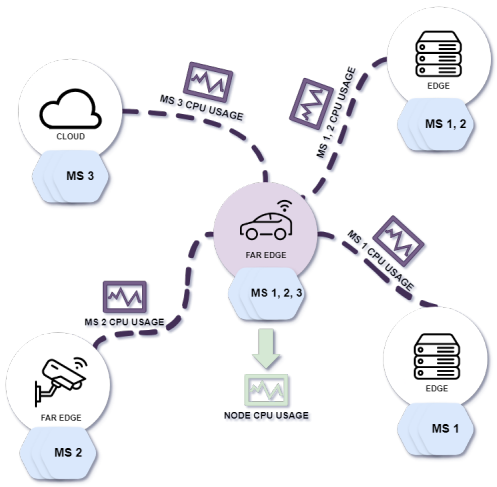


Fig. 1: Our architecture operates on a cloud-edge continuum, with nodes deployed across the cloud, edge, and far-edge. The system monitors workload and microservice performance to predict CPU usage, as illustrated by the example of a newly introduced vehicle in the network.

will provide an overview of Cloud-Edge Computing, Resource Prediction, and related work. Next, Section III will introduce the methods used for constructing the temporal graph and the GTMixer architecture. After that, Section IV will discuss the experimental setup, dataset, and the results of this work. Finally, Section V will conclude the paper.

II. Background and Related Work

A. Workload Prediction in Cloud and Edge

Effective management of edge and far-edge computing infrastructures hinges on real-time and proactive scheduling strategies. These strategies ensure seamless application performance, uphold service quality, and optimize both computational efficiency and bandwidth use, while reducing energy consumption and operational costs [3], [4]. The scientific community has developed numerous methods to address task offloading and migration within edge and cloud environments [5]–[7]. Central to these efforts is workload forecasting, especially CPU usage prediction, which is key in preventing cluster node under-utilization, reducing energy waste, and enabling proactive migration or informing the initial placement of microservices.

Kumar et al. [8] introduced a neural network optimized by an algorithm inspired by black-hole phenomena, marking a significant step forward in predicting virtual machine resource demands with improved accuracy. Further refinement in prediction methodologies is seen with Zhang et al. [9], who utilized recurrent neural networks (RNN) alongside an orthogonal experimental design to optimize influential parameters, thereby enhancing cloud workload predictions. The introduction of Bayesian uncertainty modeling by Rossi et al. [10] provided a richer understanding of demand fluctuations, acknowledging the inherent variability in cloud workloads. This complexity was further embraced by Violos et al. [11], who advanced prediction methodologies with multi-output convolutional neural networks (CNN), aiming to capture a broader spectrum of workload patterns at edge computing infrastructures. Concurrently, the GraphGRU model developed by He et al. [12] utilizes graph neural networks for resource prediction in distributed microservice environments, highlighting the potential of graph-based deep learning.

However, all these methods are static in nature, operating on fixed models that may not adapt well to the highly dynamic and volatile environments characteristic of edge computing, where topology and resources change rapidly. This limitation leads to the need for more dynamic solutions. Dynamic Temporal Graph Neural Networks, for instance, can adapt to these changes by capturing temporal and spatial dependencies, offering an improved approach to resource prediction that accounts for the volatility present in edge environments.

B. Temporal GNNs and Time Series Prediction

Graph Neural Networks utilize message-passing mechanisms to embed node and edge information, enabling the learning of complex patterns within graph data. In the landscape of discrete temporal graph neural networks, approaches such as GCRN [13] integrate graph convolutional networks with LSTMs to address temporal dynamics and spatial information, while DySAT [14] employs joint self-attention mechanisms across structural neighborhoods and temporal sequences for enhanced node representation. EvolveGCN [15] uniquely evolves GCN parameters via RNNs, focusing on the graph's dynamism rather than temporal node features.

Zeng et al. and Chen et al. challenge the complexity of deep learning models for time series forecasting. Zeng et al. [16] introduced LTSF-Linear, a simple one-layer linear model that surpasses complex Transformer-based models in long-term forecasting. Concurrently, Chen et al.'s TSMixer [17], an all-MLP architecture, effectively mixes time and feature information, demonstrating superior performance on benchmarks, including real-world retail data. Both works advocate for simpler, yet effective models in extracting insights from time series data, highlighting the potential of MLPs and linear models over more complex alternatives.

Inspired by these findings, our proposed architecture aims to synergize the spatial-temporal modeling capabilities of temporal message-passing GNNs with the efficiency and simplicity of TSMixer. By integrating these approaches, we aspire to create a framework that not only accurately captures the dynamic interdependencies characteristic of workload data in edge servers but also ensures scalability and ease of implementation.

III. Methods

The primary objective of this study is to predict the future CPU usage of physical nodes within a distributed system. To achieve this, we leverage historical data on CPU usage from these nodes, as well as the CPU usage of the various microservices that operate on them. These data points are not merely static but part of a dynamic interplay that unfolds over

time across different nodes. To capture this dynamic, we are constructing a temporal graph that dynamically profiles each microservice.

A. Dynamic Temporal Graph Construction

Given a set of physical nodes $N = \{n_1, n_2, \ldots, n_m\}$ and a set of microservices $M = \{m_1, m_2, \ldots, m_k\}$, each uniquely identified by an ID. Let U_n^t and $U_{m,n}^t$ denote the CPU usage of node n and the CPU usage of microservice instance m running on node n at time t, respectively. Time is discretized into constant intervals, indexed by t, and $WINDOW$ is the number of timesteps into the past included in the feature vector for both nodes and edges.

The complete DTG G is defined as a series of temporal slices $\{G_t\}_{t=1}^{T}$, where each G_t corresponds to the state of the graph at time t. A depiction of a temporal graph can be seen in Figure 2. Each G_t is constructed as follows:

- Each node $n \in N$ is represented as a node in G_t.
- For each microservice m_i assigned to run on node n_x at time t, an edge is added from node n_y to n_x if m_i was running on n_y at $t-1$. Formally, the edge set E_t for graph G_t is defined as:

$$E_t = \{(n_y, n_x) \mid m_i \in M, \text{ran}_{t-1}(m_i, n_y), \text{to } n_x \text{ at } t\},$$

where $\text{ran}_{t-1}(m, n)$ denotes that microservice m was running on node n at $t-1$.

Node and edge features at timestep t are defined as:

- **Node Features**: For each node n at time t, the feature vector includes the CPU usage of the physical node n for the last $WINDOW$ timesteps: $\{U_n^{t-i}\}_{i=0}^{WINDOW-1}$.
- **Edge Features**: For each edge (n_y, n_x) representing the assignment of microservice m from n_y to n_x, the feature vector includes the CPU usage trace of the microservice instance m for the last $WINDOW$ timesteps while it was running on n_y: $\{U_{m,n_y}^{t-i}\}_{i=0}^{WINDOW-1}$.

B. GTMixer Architecture

The `GTMixer` model is designed to leverage the DTG slices G_t using graph neural network principles. The model architecture is composed of N Graph Temporal Convolution (GTConv) layers followed by a projection layer that outputs the predicted features as seen in Figure 3.

1) Aggregation Layer: Each `GTConv` layer operates on nodes u by aggregating spatial information from neighboring nodes v connected through edges e_{uv}. To perform this aggregation, a Single Head Attention mechanism (Figure 4a) is used as the first step. This mechanism dynamically weighs the influence of each neighbor v based on the edge attributes e_{uv} and the destination node features. The weighted features are then aggregated with a permutation invariant function such as max, mean, or sum, denoted with $\bigoplus$. This process is represented mathematically as:

$$h_u = \bigoplus_{v \in N_u} \text{SingleHeadAttention}(x_u, e_{vu}),$$

and can be traced back to the Transformer architecture [18]. After aggregation, the spatial representation, which encapsulates the CPU consumption profiles from incoming microservices, is concatenated with the transformed representation of the source node. This node transformation is applied through a sequence that includes a linear layer, a ReLU activation, and dropout, with a residual connection to enhance the learning of subtle variations in the node features:

$$x'_u = x_u + \text{ReLU}(\text{Linear}(x_u)).$$

2) Mixer Layer: The concatenated vector is then passed through a series of `MixerLayer`, as described in the TSMixer architecture [17], which mixes features across both the node and its neighborhood to capture temporal dependencies effectively:

$$h'_u = \text{MixerLayer}(x'_u \| h_n).$$

The MixerLayer is made up of two modules: MLP_{time} and MLP_{feat}. The former is similar to the source node transformation but with an additional normalization layer at the beginning. The MLP_{feat} module combines the source node features with the aggregated representation of the incoming microservices resource usage. It has a normalization layer, followed by two linear layers, ReLU activation, and dropout. This layered approach (Figure 4b) ensures that each node's feature vector represents its attributes and integrates information from its immediate graph neighborhood over time.

3) Temporal Projection: Finally, the output from the last GTConv layer is processed through a projection layer, which linearly transforms the high-dimensional feature vector to the desired output dimension, facilitating direct prediction of the CPU usage for each node.

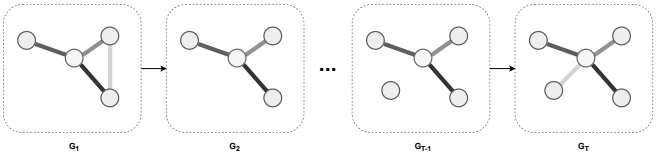


Fig. 2: A depiction of a temporal graph. Each graph slice has a unique set of node and edge features. Edges between nodes can appear and disappear at different timesteps.

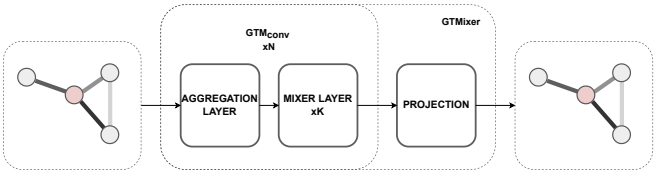
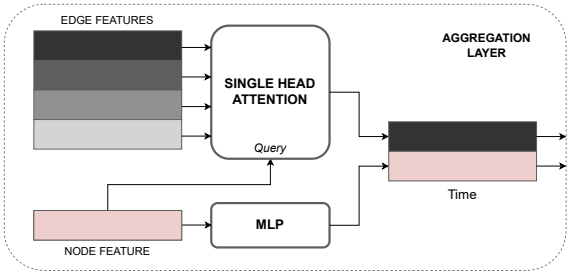
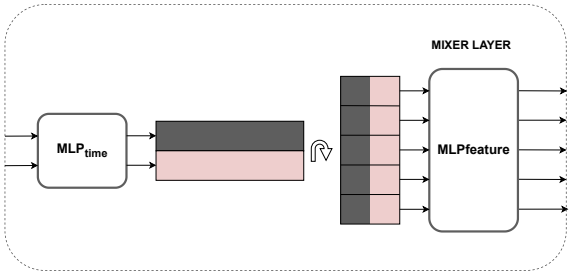


Fig. 3: The architecture of the `GTMixer` model with temporal message passing and mixing layers.



(a) Aggregation layer.



(b) Mixer layer.

Fig. 4: GTMixer architecture blocks.

This model architecture aligns with the temporal graph G_t by explicitly considering both the static and dynamic aspects of nodes and edges, thereby enabling predictions that are deeply contextualized by the historical data embedded in the graph.

IV. Experiments

A. Dataset

Due to the lack of edge trace datasets, we modified a trace dataset from an Alibaba cluster [19], which includes detailed runtime metrics for nearly 20,000 microservices. This dataset was collected from over 10,000 bare-metal nodes, documenting separate traces of CPU and memory usage over a 12-hour period in 2021. Upon analysis, we observed low variability in RAM usage across nodes, prompting us to exclude it from further experimental considerations.

To construct the temporal graph for our analysis, we utilized the node table, which included information on the CPU and RAM usage of physical nodes, and the MSResource table, which documented CPU and RAM usage traces of instances. Each instance was assigned a unique ID and a microservice ID, as well as the node ID of the physical node it was running on. In cases where a machine hosted multiple instances of the same microservice, we computed the mean CPU and RAM usage to establish the edge signals in the temporal graph.

In each timestep of our simulation, we randomly selected a percentage of nodes and erased their historical CPU utilization data to mimic the volatility inherent in edge environments. For these same nodes, we then recalculated the CPU utilization, U_n^t, by considering only the microservices incoming from the previous timestep. This approach effectively simulates a scenario in which new nodes emerge and are assigned a subset of microservices to execute.

For all experiments, we consistently selected the same 300 nodes to ensure a fair comparison. We integrated the temporal information into our prediction model by defining a $WINDOW$ of 10 timesteps and a $HORIZON$ of 3 timesteps for the prediction algorithm. To evaluate GraphGRU [12] we constructed the static relationships between nodes using the Dynamic Time Warping (DTW) algorithm as described in the paper.

TABLE I:
Optimal Hyperparameters

Hyperparameter	Value
Number of GTMConv layers	2
Number of mixer layers	4
Dropout rate	0.3
Hidden layer size	16
Normalization	2DLayerNorm
Aggregation function	Mean
Learning Rate	0.005

B. Experimental Setup and Training

Computational Environment: All experiments were conducted using a single RTX ADA6000 GPU. We employed the PyTorch framework along with the PyTorch Geometric extension.

Model Architecture: The neural network architecture comprises two GTMConv layers, each followed by four mixer layers to enhance feature integration. Each hidden layer contains 16 units and utilizes 2D Layer Normalization to ensure consistent training dynamics. A dropout rate of 0.3 is applied to prevent overfitting. The 'mean' function is used for permutation-invariant aggregation across nodes.

Training Procedure: We trained all models using the Adam optimizer [20]. An early stopping protocol was incorporated to prevent overfitting.

Hyperparameter Optimization: Hyperparameters for all models were systematically determined through a combination of random and grid search methods to ascertain the optimal model settings. The optimal hyperparameters of GTMixer are summarized in Table I.

C. Evaluation Metrics

To evaluate the performance of models predicting CPU usage in the context of the temporal graph G, we use both standard and dynamic evaluation metrics. Standard metrics include Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). The dynamic versions—Dynamic MSE (DMSE), Dynamic MAE (DMAE), and Dynamic MAPE (DMAPE)—specifically measure prediction errors for nodes that dynamically appear with no previous history.

1) Standard Metrics:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(U_n^{t_i} - \hat{U}_n^{t_i})^2$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|U_n^{t_i} - \hat{U}_n^{t_i}|$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{U_n^{t_i} - \hat{U}_n^{t_i}}{U_n^{t_i}}\right|$$

2) Dynamic Metrics: Dynamic versions of the metrics (DMSE, DMAE, DMAPE) follow the same formulas as their standard counterparts but are computed over n_{dynamic}, the number of observations for nodes that appeared dynamically without prior history.

D. Results and Evaluation

1) Model Descriptions: In our study, we evaluated the performance of four models, each briefly described below:

- **FF + Dropout:** This is a baseline model consisting of a simple feedforward neural network with dropout layers to prevent overfitting [16]. It directly predicts CPU usage from historical data without considering node interactions.
- **TSMixer:** Based on TSMixer architecture [17], this model uses an all-MLP setup to effectively integrate time and feature information for high-dimensional time-series data.
- **GraphGRU:** GraphGRU [12] combines graph neural networks with GRUs and attention mechanisms, using static relationships between nodes calculated at the start.
- **GTMixer (Ours):** GTMixer integrates GNNs with mixer layers to dynamically predict CPU usage in volatile edge environments, creating a temporal graph that reflects current and historical data interactions.

E. Performance Evaluation and Model Efficiency

The results of our study are summarized in Table II, providing a comprehensive comparison of model performance across both standard and dynamic metrics. GTMixer significantly outperforms GraphGRU, achieving improvements of 87%, 71.8%, and 97.4% in standard metrics such as MSE, MAE, and MAPE, respectively. These enhancements are even more pronounced in the dynamic metrics (DMSE, DMAE, DMAPE), highlighting GTMixer's superior ability to adapt to nodes without historical data (volatile nodes).

Comparative Analysis of Graph Models:

GraphGRU and GTMixer adopt fundamentally different approaches to handling graph dynamics. GraphGRU operates on a static graph, constructed once and used throughout the training process. This static nature limits its flexibility in highly dynamic environments, such as our dataset where 20% of the nodes randomly lose their historical data at each timestep. This limitation hampers the model's ability to propagate up-to-date information across nodes, resulting in poorer performance—even compared to non-graph models.

In contrast, GTMixer utilizes a dynamic temporal graph that is reconstructed at each timestep. By incorporating both CPU usage traces from nodes and dynamically integrating individual microservice traces, GTMixer effectively builds a dynamic profile for each microservice at every timestep. This capability allows it to handle the inherent volatility and changes in node behavior and microservice interactions within the network more effectively.

Performance of Non-Graph Models:

Non-graph models like FF + Dropout and TSMixer encounter challenges in environments with a high percentage of dynamic nodes, as shown in Figure 5. These models rely on historical trends for predictions and can only partially adapt by estimating averages when faced with new or volatile nodes. It's important to note that the dataset was sourced from an Alibaba datacenter, where the infrastructure is optimized for uniform distribution of workload. This optimization minimizes workload variations across nodes, making it more difficult to observe significant differences between non-graph and graph models. As a result, while non-graph models may appear to perform adequately on this dataset, they may not capture the complexities of real-world scenarios where workload distributions are less uniform and network loads fluctuate more frequently.

Model Efficiency:

Despite its superior accuracy, GTMixer maintains a relatively low computational footprint with only 3.5k parameters. This efficiency makes GTMixer well-suited for deployment in edge computing scenarios where computational resources are limited. A comparison of MSE and time (in seconds) for the inference of a single batch can be seen in Figure 5.

V. Conclusion

In this study, we introduced GTMixer, a novel deep-learning model designed to predict workload and resource usage in volatile edge computing nodes. Built on the DTG framework and employing graph neural network principles, GTMixer adeptly handles both spatial and temporal data, allowing for accurate anticipation of resource demands in the fluctuating landscape of edge computing. This capability is crucial for applications requiring high reliability and low latency, typical of Multi-access Edge Computing (MEC) environments. By outperforming existing state-of-the-art models in our evaluations using a new benchmark based on Alibaba's 2021 traces, GTMixer has demonstrated its effectiveness and potential to revolutionize resource management in beyond 5G networks, enhancing service reliability while minimizing operational costs.

Looking ahead, future research will focus on enhancing GTMixer's explainability by analyzing how DTGs construct resource utilization profiles for microservices over time. Additionally, testing dynamic spatio-temporal GNNs with traces from edge infrastructure or in real-world environments will be critical to deriving actionable insights for network management based on CPU usage predictions and further evaluating the usefulness of these models.

TABLE II:
EXPERIMENT RESULTS

Model	MSE	MAE	MAPE	DMSE	DMAE	DMAPE	#PARAMETERS
ff	0.0301	0.1268	0.2443	0.0315	0.1339	0.3572	143
TSMixer	0.0082	0.0721	0.1472	0.0230	0.1317	0.2666	1.1k
GraphGRU	0.0377	0.1547	3.4180	0.0306	0.1435	3.2768	167k
GTMixer(Ours)	**0.0049**	**0.0436**	**0.0882**	**0.0154**	**0.0957**	**0.2202**	3.5k

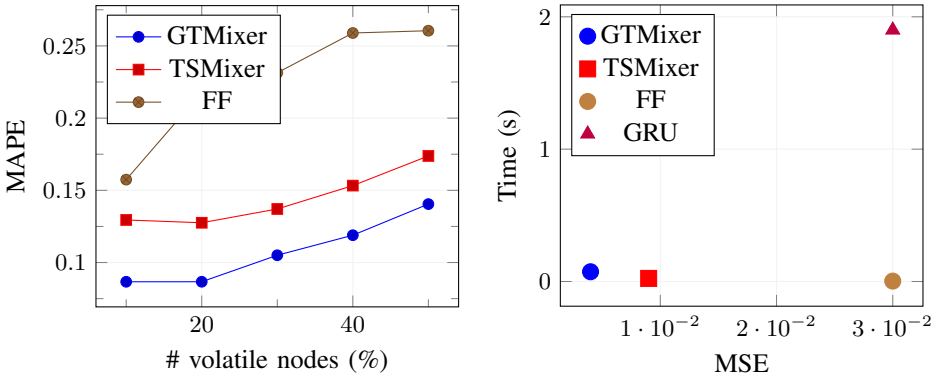


Fig. 5: Comparison of models based on MAPE scores and MSE vs. Time, illustrating the impact of volatile node percentages on MAPE and the relationship between MSE and computation time.

VI. Acknowledgment

This work was supported by the European Commission research project AC3(101093129), and partially by the Spanish Ministry of Science with ERFD funds under project PID2022-142105OB-I00.

References

[1] X. Kong, Y. Wu, H. Wang, and F. Xia, "Edge computing for internet of everything: A survey," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 23 472–23 485, 2022.

[2] C. Sergiou, M. Lestas, P. Antoniou, C. K. Liaskos, and A. Pitsillides, "Complex systems: A communication networks perspective towards 6g," *IEEE Access*, vol. 8, pp. 89 007–89 030, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218893529

[3] J. Chen, Y. Wang, and T. Liu, "A proactive resource allocation method based on adaptive prediction of resource requests in cloud computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, Feb. 2021. [Online]. Available: http://dx.doi.org/10.1186/s13638-021-01912-8

[4] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource management at the network edge: A deep reinforcement learning approach," *IEEE Network*, vol. 33, no. 3, pp. 26–33, 2019.

[5] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7635–7647, 2019.

[6] Q. Yuan, J. Li, H. Zhou, T. Lin, G. Luo, and X. Shen, "A joint service migration and mobility optimization approach for vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9041–9052, 2020.

[7] Y. Gong, H. Yao, J. Wang, L. Jiang, and F. R. Yu, "Multi-agent driven resource allocation and interference management for deep edge networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 2, pp. 2018–2030, 2022.

[8] J. Kumar and A. Singh, "An efficient machine learning approach for virtual machine resource demand prediction," *International Journal of Advanced Science and Technology*, vol. 123, pp. 21–30, 02 2019.

[9] W. Zhang, B. Li, D. Zhao, F. Gong, and Q. Lu, "Workload prediction for cloud cluster using a recurrent neural network," in *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*, 2016, pp. 104–109.

[10] A. Rossi, A. Visentin, S. Prestwich, and K. N. Brown, "Bayesian uncertainty modelling for cloud workload prediction," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, 2022, pp. 19–29.

[11] J. Violos, T. Pagoulatou, S. Tsanakas, K. Tserpes, and T. Varvarigou, "Predicting resource usage in edge computing infrastructures with cnn and a hybrid bayesian particle swarm hyper-parameter optimization model," Cham, pp. 562–580, 2021.

[12] H. He, L. Su, and K. Ye, "Graphgru: A graph neural network model for resource prediction in microservice cluster," pp. 499–506, 2023.

[13] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," 2016.

[14] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dynamic graph representation learning via self-attention networks," 2019.

[15] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," 2019.

[16] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 9, 2023, pp. 11 121–11 128.

[17] S.-A. Chen, C.-L. Li, N. Yoder, S. O. Arik, and T. Pfister, "Tsmixer: An all-mlp architecture for time series forecasting," 2023.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[19] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 412–426.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106