Forecasting Individual NetFlows using a Predictive Masked Graph Autoencoder

Georgios Anyfantis
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
georgios.anyfantis@upc.edu

Pere Barlet-Ros
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
pere.barlet@upc.edu

***Abstract*—In this paper, we propose a proof-of-concept Graph Neural Network model that can successfully predict network flow-level traffic (NetFlow) by accurately modelling the graph structure and the connection features. We use sliding-windows to split the network traffic in equal-sized heterogeneous bidirectional graphs containing IP, Port, and Connection nodes. We then use the GNN to model the evolution of the graph structure and the connection features. Our approach shows superior results when identifying the Port and IP to which connections attach, while feature reconstruction remains competitive with strong forecasting baselines. Overall, our work showcases the use of GNNs for per-flow NetFlow prediction.**

***Index Terms*—Graph Neural Networks, NetFlow Forecasting, Graph Masked Autoencoder**

I. Introduction

Forecasting network traffic can be used for traffic engineering, routing, improving resource allocation, and service orchestration by anticipating future patterns that are not visible at the aggregate level [1].

A substantial body of research on the prediction of network traffic focuses on aggregated metrics [2], such as link utilisation over time. However, there has been limited research on predicting individual NetFlows [3]. This is not surprising due to the increased complexity of the forecasting problem [3]. Flows tend to arrive at large volumes, combine both numerical and categorical data, and exhibit periodic changes, making them very difficult to model correctly. Moreover, early work on fine-grained prediction noted that per-flow forecasting was impractical [1].

However, per-flow forecasting can be very valuable as it preserves a lot of information that aggregated forecasting removed [1]. This can help predict potential bottlenecks and resource demand, support more fine-grained network engineering decisions, and improve the modelling of distributed applications. Finally, recent work on flow-level simulation indicates that aggregate models are insufficient for tasks that require fine-grained information [4].

Previous work on ML based approaches for network traffic has suffered from the smoothing problem on aggregated metrics as aggregated smoothing tends to hide micro-bursts [5]. These micro-bursts can contain important information about the network that is lost.

Graph Neural Networks (GNNs) have been found to be able to solve the stagnation problem observed when modelling networks by leveraging the graph structure [6]. This is not possible in traditional models, as they cannot leverage the relationship between individual flows. GNNs are Neural Networks that use both the node features and the graph structure to capture relational information [7].

Per-flow forecasting is an inherently relational task. A NetFlow is not an isolated observation, but rather a part of a larger communication pattern [8]. As such, modelling NetFlows solely as tabular data may miss important relational information [9]. GNNs are ideal for this task, as they capture the network interactions and better model the changing nature of the network, making GNNs the ideal choice for per-flow prediction.

IPFIX and NetFlows remain a widely used and practical format to capture network telemetry for monitoring. They provide a standardised representation of communication behaviour inside the network without requiring full payload inspection [10]. This makes them an ideal format for use for forecasting tasks.

In this paper, we employ Graph Neural Network (GNN) for fine-grained per-flow prediction using the NetFlow format. Our work is a proof-of-concept approach showcasing the capability of GNNs for per-flow prediction. We compare our proposed solution with other strong Machine Learning forecasting methodologies. Our results indicate that GNNs are very well suited for per-flow prediction and showcase a clear structural advantage and competitive feature reconstruction compared to the baselines employed in this paper.

II. Problem Description

Flow prediction is usually tackled as a time-series problem [2]. In this paper, we are looking at predicting flows as a sliding window one-step forecasting problem. One-step forecasting problems look at taking the current step and predicting the next step without any input from the previous steps. In our approach, we are taking the current step that represents the network traffic and predict the next step with the future network traffic.

In the per-flow prediction, we slice the traffic into non-overlapping windows of equal lengths. Then we feed the

current window to the model and we try to predict the next sequence of flows.

To evaluate the efficacy of GNNs for per-flow prediction, we have implemented some strong sequence-prediction backbones. These methodologies share the same encoder and decoder architecture, but we switch the backbone to implement each architecture.

The Long-Short-Term Memory (LSTM) model [11] is a well-known recurrent model architecture that is widely used for forecasting and modelling [12]. The architecture works by ingestion of the input sequentially and updating its internal hidden state through gated updates, helping the model learn temporal dependencies over long horizons [11]. LSTMs have been extensively researched and are known as a standard forecasting architecture [12].

Temporal Convolution Network (TCN) is a forecasting architecture that switches recurrence with causal one-dimensional convolutions [13]. In their standard form, TCNs use causal convolutions with dilation and residual connections, allowing the receptive field to grow while preserving temporal order. This makes them an attractive alternative to standard recurrent architectures for modelling long-range dependencies. In the original paper, TCN was found to perform better than traditional approaches in long-range dependency modelling [13] and a recent survey indicates that TCN is widely used in time-series forecasting [14].

Transformers are attention-based models that were originally built as encoder-decoder architectures without recurrence or convolutions [15]. The Transformer's main strength is the use of the self-attention mechanism for capturing and modelling long-range dependencies. In time series forecasting, Transformer based models have been proposed and have been successfully applied to time series forecasting and tasks [16].

DLinear is a simple linear forecasting architecture [17]. It works by decomposing the input into a trend component and a seasonal or remainder component using a moving-average decomposition. Then it applies separate one-layer linear projections to the two components and combines them to a final projection. It is a more recent model than LSTM [11], TCN [13] and Transformers [15] it has been found to be a very good baseline used for benchmarking in long-term time-series forecasting.

Graph Neural Networks are a class of Neural Networks that uses structured graph data to learn representations by propagating and aggregating information across connected graph nodes [18]. GraphSAGE is one of the most well-known GNN algorithms used in inductive formulations, and it works by aggregating features from local neighbourhoods, allowing the model to capture both the structure and the attributes of the graph [19].

III. Methodology

A. Dataset and Graph Creation

For this paper, we used the UNSW-NB15 dataset [20]. We chose to use this dataset as it is a widely used dataset with individual flows. For this paper, we chose to extract NetFlows V9 [21] from raw PCAP files. For extraction, we used NFStream [22], which is a lightweight Python library that can extract NetFlows from raw PCAP files. We followed a similar pre-processing and graph creation approach as we had followed in our previous paper [23].

For data pre-processing, we use L2 normalisation for numerical data. We chose this normalisation as it is very efficient and avoids the requirement of pre-training an encoder and having to re-tune to account for distribution drifts. For categorical data, such as ports and protocols, we use One-Hot Encoding (OHE).

For the OHE, we used pre-determined values for the most important categories. The schema can be seen in Table I. Due to the nature of the UNSW-NB15 dataset, we discarded the IP version since all traffic is IPv4.

TABLE I
One-Hot Encoding (OHE) Mapping Schema

Feature	OHE Category	Original Values / Logic
Ports	Port 0	0
	Port 53	53
	Port 123	123
	Port 443	443
	Well-Known	$0 \leq x \leq 1023$ (Excl. specific ports)
	Registered	$1024 \leq x \leq 49151$
	Dynamic/Private	$49152 \leq x \leq 65535$
Protocols	ICMP	1
	TCP	6
	UDP	17
	Others	All other protocol values

Regarding graph building, we chose to use fixed sized sliding windows. The data is split into non-overlapping windows of length 512. We chose this length because it has been well documented that the baseline methods can handle this sequence length very well. The graphs were constructed as heterogeneous. We are using IP nodes that have placeholder features initialised to ones, 8 unique Port nodes with initialised placeholder features to ones, and the connection nodes with connection features. The port nodes represent the OHE categories, and their features were initialised to ones as there was no performance enhancement when richer port features were used. A representation of how NetFlows are represented in our graphs can be found in Figure 1.

For the baselines, we have followed the same pre-processing and sliding window splitting. The IPs and Ports are mapped to unique IDs that the model then processes. The ports are mapped using the OHE categories.

The whole pre-processing step was implemented using NumPy [24].

In terms of dataset splits, we chose to allocate 70% of the dataset in the training set, 20% of the dataset in the validation set, and 10% for the testing part. We used *train_test_split* from Scikit-Learn [25] to create the splits. The splits are random and do not use any of the dataset labels. Since only the current window is used for the prediction, we decided that this approach is appropriate since there is no temporal overlap

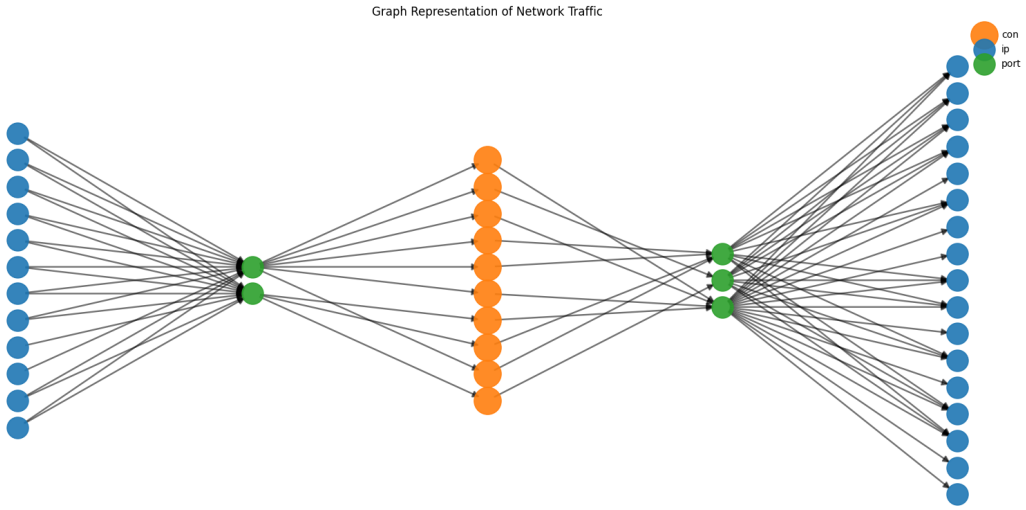

Fig. 1. A graph representation on how the NetFlows are represented in our Graphs. This is a sampled subset of 10 connections.

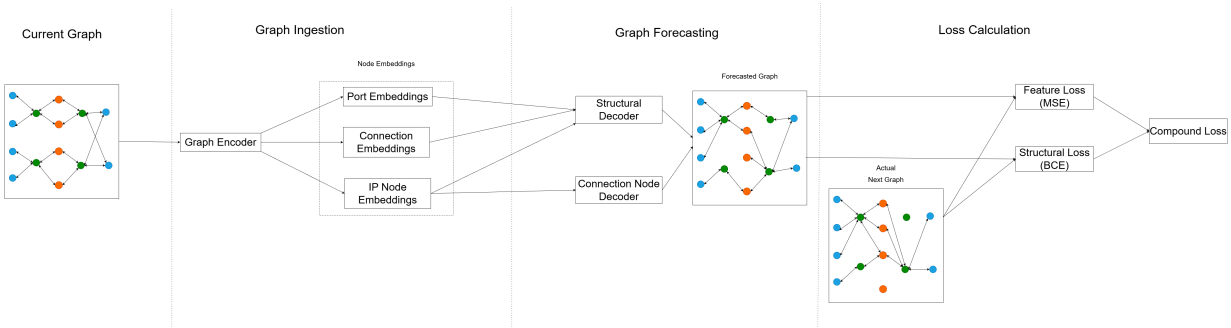

Fig. 2. A representation of our architecture and its training procedure.

in the training as each step is seen individually and no past steps are fed into the model.

B. Model Architecture

Our proposed model has been inspired by our previous work on AutoGraphAD [23]. The model works by mapping the current graph into the future graph. For this task, we employ a Graph Autoencoder [26].

For our proposed model, we have built on our previous work of using masked Graph Variational Autoencoders (GVAE) [23]. Our architecture can be seen in Figure 2.

For this paper, we are using a simple Graph Autoencoder (GAE) that adapts the Graph Masked Autoencoder Approach (GraphMAE) [27] and heterogeneous graph mask autoencoders (HGMAE) [28]. GraphMAE uses a learnable embedding to perform masking, similar to BERT [29]. We are using the same masking technique as proposed by HGMAE and GraphMAE but we do not perform any masking on the edges. We are focusing on reconstructing the masked nodes and all the graph edges.

We are using random edge dropping as part of the training to force the model to learn non-trivial solutions and better capture the structural information of the graph [28]. Thus, allowing for better reconstruction.

Our autoencoder uses GraphSAGE [19] as it is a proven algorithm that works well with unseen graphs.

For our node feature decoder, we are using a simple Multi-Layer Perceptron (MLP). We have chosen to use this architecture because it is simple and has proven to be very successful when used for GAE [28]. Moreover, the use of an MLP compared to a GNN decoder removes the need of needing the future graph's edge index, something that should not be relied on in a forecasting problem.

For structural reconstruction, we are using an enhanced Dot Product [23]. The enhanced Dot Product is based on the original Dot Product [26] but is enhanced by using learnable weights to better learn the structural information of the graph.

For our backpropagation loss, we use composite loss. It is defined by the weighted sum of the Feature Loss and the Structural Loss. The user can assign importance to the Structural Loss by setting the balancing hyperparameter α. The loss can be seen in Equation 1.

$$L_{Total} = \alpha * StructLoss + (1 - \alpha) * FeatLoss \quad (1)$$

Our new model differs from AutoGraphAD as it does not use a GVAE architecture and during training only reconstructs the masked features. Additionally, it does not employ masking on the edges as they are not static between the current and next graph. Furthermore, we use a weighted sum loss instead of the highly configurable loss used in AutoGraphAD. Finally, AutoGraphAD focuses on reconstructing the inputted graph, whereas our proposed GNN focuses on predicting the next graph.

C. Baseline Architecture

For our baseline architecture, we are using a traditional masked autoencoder [30]. We have chosen this since it is the closest to our Masked GAE and, for modelling problems, Masked Autoencoders are ideal. For masking, for categorical data we use a separate token while for numerical features we use a learnable embedding similar to how BERT works [29].

Our baselines use a unified backbone architecture, which means that all baselines share the encoder and decoder mechanism. For the encoder, we are using individual embedding layers [31] to map the IP address octets. The embedding layers use learnable tables to better map values into fixed sized vectors that are then combined with the rest of the input features to form the input sequence.

Additionally, all baselines share the same decoding architecture. For a decoder, we employ an MLP with two heads. One that reconstructs the IP and Port information, and another that reconstructs the rest of the flow features. This architecture mirrors the flow reconstruction approach that is employed by our GNN. Only the algorithm used for future forecasting is swapped out based on the baseline that we are using. As mentioned in Section II, we use DLinear [17], TCN [13], LSTM [11], and Transformer [15] architectures.

D. Libraries Used

We implemented the Baselines using PyTorch [32] and PyTorch Geometric [33] for GNN. We used PyTorch Lightning [34] to allow faster code development and better reusability. Torchmetrics [35] was used for the evaluation metrics.

To run our experiments, we used a server with an Nvidia RTX 3090 with 24 GB of VRAM. The CPU of the server is an AMD Ryzen 3950X, 16-Core Processor. The server came with 64 GB of RAM and its operating system was Ubuntu 22.04.4 LTS.

IV. Experimental Results

A. Hyperparameter Tuning

To evaluate and find the optimal hyperparameters, we have employed Optuna [36]. Optuna is a framework that allows for an efficient and informed hyperparameter space search. It provides the user with an array of sampling and pruning strategies that allow the framework to perform informed hyperparameter sampling using a user defined metric.

In our case, both the baselines and the GNN used Successive Halving [37] as a pruning strategy. Successive Halving is a hyperparameter optimisation strategy that tries to identify optimal hyperparameter settings by aggressively pruning underperforming trials. It evaluates different trials at different "rungs," where each run starts with minimal resource allocation, such as epochs. At each subsequent rung, only the highest performing trials, determined by a reduction factor η, are allowed to progress and use more resources. Allowing resources to be redirected to the most promising trials.

For the sampler, we have used Optuna's TPESampler. TPESampler is the implementation of the Tree-Structured Parzen Estimator (TPE) [38]. TPE is a Bayesian optimisation method that uses Sequential Model-Based Optimisation instead of Gaussian Processes. Instead of modelling how the hyperparameters affect the score, TPE models the probability density of the hyperparameters that produces the best score $l(x)$ and the density of the hyperparameters that produces the worse score $g(x)$. TPE tries to maximise the ratio between the two densities by identifying the highest Expected Improvement (EI).

The Optuna TPESampler and SuccessiveHalving settings can be found in Table II. For our optimisation, we used the compound metric derived during the validation step in the training. The compound metric is a weighted metric that combines different metrics to create a single metric for which Optuna will focus on optimising. This metric incorporates important information about different aspects of the model. The compound metric can be seen in Equation 2.

$$CompScore = 0.25 * Acc + 0.25 * AUROC + 0.5 * (1 - MAE) \quad (2)$$

TABLE II
Hyperparameter Optimization Configuration

Component	Parameter	Value
Sampler (TPE)	Seed	42
	Startup Trials ($n_{startup}$)	10
	EI Candidates (n_{ei})	24
	Multivariate	True
	Group	True
Pruner (SHA)	Minimum Resource (R_{min})	8
	Reduction Factor (η)	3
	Min. Early Stopping Rate	0
	Bootstrap Count	2
Training Parameters	Training Epochs	75
	Number of DataLoader Workers	8
	Batch Size	64
	Pin Memory	True
	Persistent Workers	True
	Window Length	512
	Window Stride	512
	Gradient Accumulation Steps GNN	2
	Gradient Accumulation Steps Baselines	1

B. Results

Regarding the metrics used in this paper, we chose to use MSE and MAE for the numerical feature reconstruction,

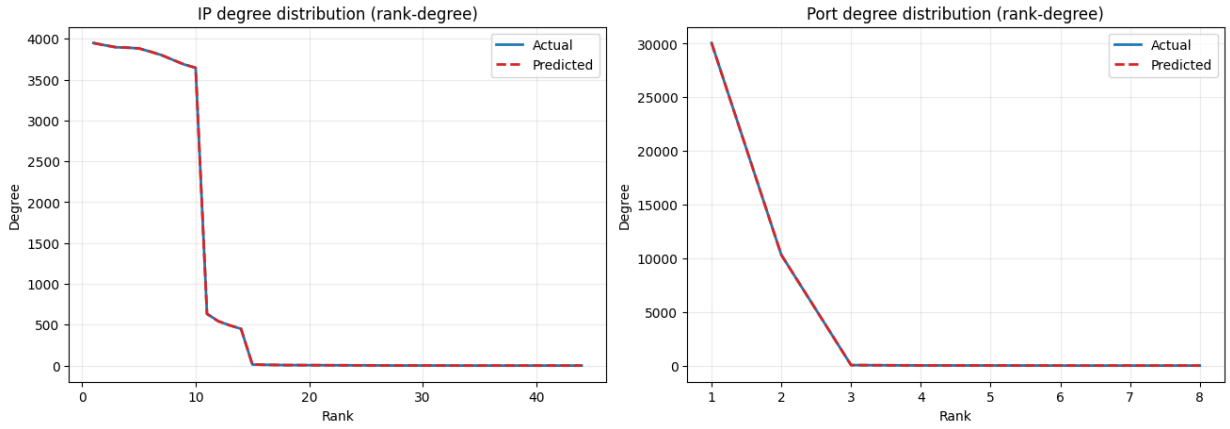

Fig. 3. The actual ranked connectivity degree of the IP nodes and the forecasted degree by the GNN. Ranked degrees showcase the IPs and Ports that have the highest to the lowest connectivity.

TABLE III
MODEL PERFORMANCE: FEATURE RECONSTRUCTION VS. STRUCTURAL PREDICTION

Model	Feature Reconstruction		Structural Reconstruction		
	MAE (↓)	**MSE** (↓)	**Accuracy** (↑)	**AUROC** (↑)	**Precision** (↑)
GNN	**0.0647**	0.0196	**87.9%**	**95.1%**	**87.8%**
LSTM	0.0650	**0.0194**	16.4%	41.1%	19.2%
TCN	0.0650	0.0195	16.4%	39.3%	15.4%
Transformer	0.0660	0.0196	16.2%	41.1%	15.5%
DLinear	0.0660	0.0197	16.4%	39.1%	15.7%

and for the structural data, we are using Macro metrics for Accuracy, AUROC, and Precision. For the evaluation, we are choosing the Baseline Models and the GNN that have achieved the highest compound score and the test split of the dataset.

Looking at the results in Table III, we can see that the proposed GNN outperforms all baselines when it comes to reconstructing the IP and Port information. This is because of the GNN's ability to model structural dependencies in the Graphs. This allows for better modelling of the IP and Port connections, which is necessary to identify where different NetFlow flows occur. In the case of the baselines, we can clearly see that no baseline is able to identify how each NetFlow is connected to the correct IP address and Port, with their Accuracy being at 16% for the baselines and 87.5% for the GNN. The same can be seen in the case of AUROC and Precision metrics where the GNN clearly outperforms the baseline models, again demonstrating the advantage of graph representation for forecasting tasks. This can also be seen in Figure 3, which shows the degree of connectivity of the IP and Port nodes. The nodes are ranked from the most connected to the least connected and show how many edges originate from them. Our proposed GNN can be seen to perform exceptionally well by matching the degree connectivity of the actual nodes.

Looking at the reconstruction of features, we can see a similar approach; overall, the GNN is very closely correlated to the LSTM baseline as seen in Table III. LSTM only slightly outperforms the GNN at the MSE but on MAE GNN remains the best approach. This indicates that the GNN seems to be performing exceptionally well in the majority of the data, yet it struggles when it comes to outliers, as seen by the MSE error, whereas LSTM is performing better at the outliers, but tends to be less accurate at the majority of the data.

V. CONCLUSION AND FUTURE WORK

The proposed GNN showcases the capabilities of using graphs to model NetFlows at a fine-grained level compared to previous research that focused on aggregated metrics. Our model is able to outperform existing baselines and showcases that GNNs can successfully model NetFlow connectivity and their features.

Our future work will focus on improving training and evaluating performance on a larger set of datasets, as well as supporting variable graph sizes, which is necessary to better model the non-static nature of network traffic.

ACKNOWLEDGEMENTS

This work was supported by Grant PCI2023-145974-2 funded by MICIU/AEI/10.13039/501100011033 and cofunded by the European Union (GRAPHS4SEC project). This work is also supported by the Catalan Institution for Research and Advanced Studies (ICREA Academia). OpenAI Codex was used to help debug and develop the code. It served mainly as a search engine for what was causing the bugs and as a recommendation engine to check which libraries

and their methods are the most suitable and optimal during development.

REFERENCES

[1] B. Pfülb, C. Hardegen, A. Gepperth, and S. Rieger, *A Study of Deep Learning for Network Traffic Data Forecasting*, p. 497–512. Springer International Publishing, 2019.

[2] G. O. Ferreira, C. Ravazzi, F. Dabbene, G. C. Calafiore, and M. Fiore, "Forecasting network traffic: A survey and tutorial with open-source comparative evaluation," *IEEE Access*, vol. 11, pp. 6018–6044, 2023.

[3] P. Jahnke, E. Stapf, J. Mieseler, G. Neumann, and P. Eugster, "Towards fine grained network flow prediction," 2018.

[4] C. Li, A. A. Zabreyko, A. Nasr-Esfahany, K. Zhao, P. Goyal, M. Alizadeh, and T. Anderson, "m4: A learned flow-level network simulator," 2025.

[5] J. François, A. Clemm, D. Papadimitriou, S. Fernandes, and S. Schneider, "Research Challenges in Coupling Artificial Intelligence and Network Management," Internet-Draft draft-irtf-nmrg-ai-challenges-05, Internet Engineering Task Force, Mar. 2025. Expired Internet-Draft.

[6] R. Wang, J. Zhao, H. Zhang, L. He, H. Li, and M. Huang, "Network traffic analysis based on graph neural networks: A scoping review," *Big Data and Cognitive Computing*, vol. 9, no. 11, 2025.

[7] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[8] R. Xu, G. Wu, W. Wang, X. Gao, A. He, and Z. Zhang, "Applying self-supervised learning to network intrusion detection for network flows with graph neural network," *Computer Networks*, vol. 248, p. 110495, 2024.

[9] D. Pujol-Perich, J. Suárez-Varela, A. Cabellos-Aparicio, and P. Barlet-Ros, "Unveiling the potential of graph neural networks for robust intrusion detection," 2021.

[10] P. Aitken, "Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information," tech. rep., Cisco Systems, Inc., Sept. 2013.

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

[12] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia CIRP*, vol. 99, pp. 650–655, 2021. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

[13] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[14] X. Kong, Z. Chen, W. Liu, K. Ning, L. Zhang, S. Muhammad Marier, Y. Liu, Y. Chen, and F. Xia, "Deep learning for time series forecasting: a survey," *Int. J. Mach. Learn. Cybern.*, vol. 16, pp. 5079–5112, Aug. 2025.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[16] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey," 2023.

[17] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?," 2022.

[18] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[19] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[20] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, 2015.

[21] B. Claise, "Cisco systems netflow services export version 9," tech. rep., Cisco, 2004.

[22] Z. Aouini and A. Pekar, "Nfstream: A flexible network data analysis framework," *Computer Networks*, vol. 204, p. 108719, 2022.

[23] G. Anyfantis and P. Barlet-Ros, "Autographad: A novel approach using variational graph autoencoders for anomalous network flow detection," 2025.

[24] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016.

[27] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, "Graphmae: Self-supervised masked graph autoencoders," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, (New York, NY, USA), p. 594–604, Association for Computing Machinery, 2022.

[28] Y. Tian, K. Dong, C. Zhang, C. Zhang, and N. V. Chawla, "Heterogeneous graph masked autoencoders," 2023.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[30] K. He, X. Chen, S. Xie, Y. Li, P. Doll'ar, and R. B. Girshick, "Masked autoencoders are scalable vision learners," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15979–15988, 2021.

[31] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[33] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019.

[34] W. Falcon and The PyTorch Lightning team, "Pytorch lightning," March 2019.

[35] N. S. Detlefsen, J. Borovec, J. Schock, A. H. Jha, T. Koker, L. Di Liello, D. Stancl, C. Quan, M. Grechkin, and W. Falcon, "Torchmetrics - measuring reproducibility in pytorch," *Journal of Open Source Software*, vol. 7, no. 70, p. 4101, 2022.

[36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[37] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," in *Proceedings of Machine Learning and Systems* (I. Dhillon, D. Papailiopoulos, and V. Sze, eds.), vol. 2, pp. 230–246, 2020.

[38] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.