Can Semi-Supervised Learning Improve Prediction of Deep Learning Model Resource Consumption?

Karthick Panner Selvam
University of Luxembourg, SnT
Luxembourg
karthick.pannerselvam@uni.lu

Mats Brorsson
University of Luxembourg, SnT
Luxembourg
mats.brorsson@uni.lu

Abstract

With the increasing computational demands of Deep Learning (DL), predicting training characteristics like training time and memory usage is crucial for efficient hardware allocation. Traditional methods rely solely on supervised learning for such predictions. Our work integrates a semi-supervised approach for improved accuracy. We present TraPPM, which utilizes a graph autoencoder to understand representations of unlabeled DL graphs, then combined with a supervised graph neural network training to predict the metrics. Our model significantly surpasses standard methods in prediction accuracy, with MAPE values of 9.51% for training step time and 4.92% for memory usage. The code and dataset are available at https://github.com/karthickai/trappm

1 Introduction

The rapid evolution of Deep Learning (DL) and the increasing complexity of its models necessitate accurate predictions of training characteristics, such as memory consumption and training time. Predicting these attributes optimizes hardware utilization and cost-effectiveness. However, predicting the training characteristics of DL models remains a significant challenge, often involving substantial trial and error Menghani [2023]. Earlier efforts predominantly used supervised MLPs and Graph Neural Networks (GNNs) for this task Justus et al. [2018], Yu et al. [2021], Gao et al. [2023], Panner Selvam and Brorsson [2023], Bai et al. [2022], Gianniti et al. [2018], Kaufman et al. [2021], Dudziak et al. [2020], Liu et al. [2022]. Yet, an untapped reservoir of potential exists in the use of unlabeled DL models.

We introduce the Training characteristics Performance Predictive Model (TraPPM), a novel semi-supervised learning framework that leverages the power of unsupervised GNN; TraPPM derives intricate graph embeddings from an unlabeled DL model graph. And combine the embedding with DL static features to train the GNN-based regressor model using a labeled dataset to predict the training characteristics without running it on target hardware. In a rigorous comparative assessment against state-of-the-art baselines, including supervised GNN, MLP, and GBoost, TraPPM exhibits superior performance, achieving a remarkable 910 MB RMSE and 4.92% MAPE for memory and 23 ms RMSE and 9.51% MAPE for step-time prediction. This superior performance underscores the efficacy of harnessing unlabeled data for performance prediction. Furthermore, our comprehensive dataset, encompassing 7,536 labeled graphs and 25,053 unlabelled graphs from various DL model families, presents a substantial contribution to the community, paving the way for future research in performance prediction and optimization.

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA.

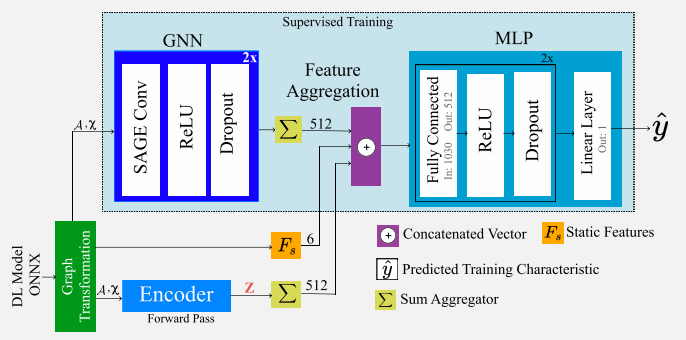


Figure 1: Training a GNN regressor using MSE loss to minimize the actual y vs. predicted $\hat{y}$.

2 Related Work

Performance prediction of DL models is a burgeoning field. Qi et al. [2017] use a analytical approach to estimate the training time of DL models, layer by layer. Gao et al. [2020] also used an analytical model to predict the memory consumption for the training DL model. Bouhali et al. [2021] used an MLP-based regressor that used input features such as trainable parameter count, memory size, and input size to predict the execution time. Nevertheless, the traditional MLP method could have been more effective due to its limited understanding of the DL layers. Justus et al. [2018] and Gianniti et al. [2018] used the layer-by-layer technique proposed by Qi et al. [2017] but used an MLP-based regression model. Other researchers Sponner et al. [2022], Lu et al. [2021], Velasco-Montero et al. [2020], Cai et al. [2017] also used the same layerwise approach to predict the characteristics of the DL model. Yu et al. [2021] employed a wave-scaling method for estimating the training step time of the deep learning model on a GPU. They also used the layerwise approach. However, this technique necessitates the availability of a GPU to facilitate the prediction. On the other hand, researchers used a GNN instead of MLP in a layerwise approach to predict the performance of the DL model Kaufman et al. [2021], Dudziak et al. [2020]. The layerwise approach did not capture the DL model network topology, and therefore prediction accuracy is sub-optimal Liu et al. [2022]. To solve the above problem, Gao et al. [2023], and other researchers, Liu et al. [2022], Bai et al. [2022], Panner Selvam and Brorsson [2023], have employed GNNs to represent a complete DL graph as input to predict both training and inference characteristics. The majority of prior studies utilized supervised techniques for DL model performance prediction, neglecting the vast pool of unlabelled DL model data. Our innovative approach, TraPPM, bridges this gap using a semi-supervised learning paradigm, enhancing prediction accuracy by harnessing unlabelled data.

3 Methodology

Given a DL model M with operations $O = \{o1, o2, ..., on\}$, we transform M (in ONNX format) into a graph G compatible with PyTorch Geometric (PyG). In G, nodes represent M's operations stored in the node feature matrix X, while A captures directed relationships. Specifically, $G = (X, A)$ where $X = O$ and $A[i][j] = 1$ if a directed edge exists from o_i to o_j, else $A[i][j] = 0$. If M is labeled, we incorporate a target vector Y into PyG data. For each node v in the DL model graph, we define an attribute vector A_v as: $[\mathbf{E}_O(v), I_v, \mathcal{O}_v, \text{Mac}_v, P_v, M_v]$. Here, $\mathbf{E}_O(v)$ is a one-hot encoded vector of length $|O|$, where $|O| = 98$, surpassing the previous work supported only 32 operators Liu et al. [2022]. The vectors I_v and $\mathcal{O}_v$, each of length 6, encapsulate the input and output shape, respectively, with an extension to consider 3D convolution. The attributes Mac_v, P_v, and M_v symbolize the MAC, parameters, and memory of node v, respectively. Thus, our node feature vector n has a dimensionality of 113, offering a more exhaustive representation. To the best of our knowledge, this is the first work to incorporate 2D and 3D convolutions, alongside transformer-based architectures, into node features. This advancement distinguishes our approach from prior studies that were limited to 2D convolutions. To exploit the rich information in unlabeled data, we employ a Graph Autoencoder (GAE). The encoder ingests node feature matrix X of dimension $[\#nodes, 113]$ and edge indices A. It comprises four consecutive SAGEConv layers. The first layer expands the feature dimensions to $2c$, where $c = 512$, and subsequent layers refine

them to the target embedding dimension c. The transformation within each convolutional layer is: $T(X, A, c) = \text{Dropout}(\text{ReLU}(\text{BatchNorm}(SAGEConv(X, A, c))), 0.5)$. The architecture we've designed is adept at generating embeddings that holistically encapsulate the nuanced attributes of DL models. Following the encoding process, the decoder aims to reconstruct the adjacency matrix using the formula $\hat{A} = \sigma(X_4 X_4^T)$. The primary goal of our GAE is to minimize the deviation between the true adjacency matrix A and $\hat{A}$, derived from the latent space representation z. The discrepancy between these matrices is measured using the Binary Cross Entropy (BCE) loss, defined as:

$$L_{\text{BCE}} = -\log(\hat{A}(z, i_{\text{pos}}, j_{\text{pos}}) + \epsilon) - \log(1 - \hat{A}(z, i_{\text{neg}}, j_{\text{neg}}) + \epsilon)$$

In this equation, $\hat{A}$ denotes the predicted adjacency matrix. The terms $i_{\text{pos}}, j_{\text{pos}}$ signify the indices of positive edges, while $i_{\text{neg}}, j_{\text{neg}}$ correspond to negative edges, obtained through negative sampling. To ensure stability during the computation of logarithms, we used a small constant $\epsilon = 1 \times 10^{-15}$. The essence of this loss metric lies in its ability to guide the GAE towards accurately reflecting the original graph structure. Post-training, the encoder's weights are freezed, yielding embeddings for ensuing supervised tasks. The primary goal of TraPPM is to predict the DL training characteristics, specifically memory usage μ (MB) and step time τ (ms). The overview of TraPPM is shown in Figure 1. From the input graph G we extract static features F_s. It encompass the batch size B, the total number of nodes N_t, the total number of edges E_t, total MAC operations (MAC_t), total parameters (P_t), and total memory (M_t). The values N_t and E_t are directly extracted from G, while the values MAC_t, P_t, and M_t are obtained using the ONNX tool. Supervised learning consists of three components. Initially, the GNN component processes the graph, utilizing two SAGEConv layers, each complemented by a ReLU activation and a 0.05 dropout. This results in embeddings E_{GNN} with dimensions [1, 512] after sum aggregation. Concurrently, embeddings from the GAE (E_{GAE}) are reduced to a similar dimensionality after sum aggregation. The feature aggregation component then fuses these embeddings with static features F_s to form an enriched vector $V = E_{\text{GAE}} \oplus E_{\text{GNN}} \oplus F_s$, which spans [1, 1030]. The MLP component subsequently processes V through two FC layers, each accompanied by ReLU activations and a 0.05 dropout, followed by a final linear layer yielding a singular output Y. Using the Mean Squared Error (MSE) as a loss function and the Adam optimizer for updates, two distinct models M_1 for μ and M_2 for τ are trained, embodying our comprehensive approach to predicting DL model training characteristics.

4 Experiments and Results

For our TraPPM experiments, we constructed datasets differentiated as unsupervised (D_U) and supervised (D_S). D_U consists of 25,053 unlabeled DL models across 11 families. D_S, derived from D_U, contains 7536 labeled DL models. Detailed environmental setup and data collection can be found in Section A.1 and A.2.

The first phase of the TraPPM experiment involves training the GAE. Initially, we considered the Masked Graph Autoencoder technique, as presented in Hou et al. [2022] study. This method masks random node features and attempts to reconstruct them, facilitating graph representation learning. However, our node features, largely sparse due to one-hot encoding, did not align well with this strategy. As a result, we turned to the classical GAE, which proved to be a better fit for our needs. The GAE model was developed using the PyG Library. For training, we set Adam optimizer with a lr $= 5 \times 10^{-4}$, $\beta = (0.9, 0.999)$, and $\epsilon = 1 \times 10^{-8}$. To train the GAE, we utilized an D_U, for a total of 400 epochs. Finally, we have achieved a L_{BCE} of 0.9291. The core part of the experiment involves training the TraPPM model. We used the PyTorch library to create the TraPPM model. We used a D_S collected from A100 GPU to train the model. We partitioned our dataset according to a 70:30 ratio for each model family. Specifically, 70% of the data was used for training and 30% for testing. We adopted a Monte Carlo validation approach. To ensure robustness and reliability in our results, we employed five distinct seeds: $\mathcal{S} = \{1337, 1338, ..., 1341\}$. By utilizing these seeds, we generated five different dataset splits and subsequently averaged the results to derive a more comprehensive performance evaluation. During the training process, we utilized the Adam optimizer with a lr $= 1 \times 10^{-3}$, $\beta = (0.9, 0.999)$, and $\epsilon = 1 \times 10^{-8}$. We assessed the performance of the TraPPM model by comparing it with baseline models. Detailed baseline setup explained in Section A.4. Both the TraPPM model and the baselines were trained using a D_S, with a specific focus on predicting μ and τ. We trained the TraPPM, NNLQP, and MLP models for 100 epochs,

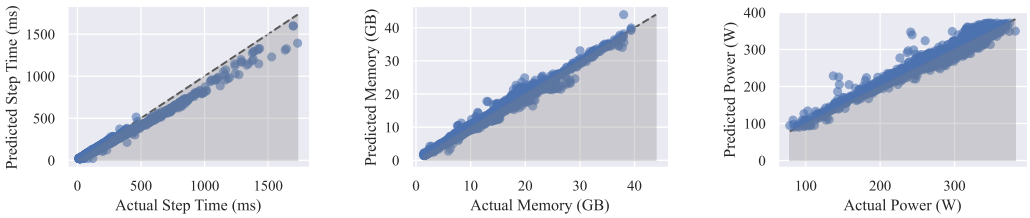


Figure 2: Comparison of actual values with predictions from TraPPM on the test set.

repeating the process five times using different seeds as mentioned above to perform a fair comparison with baseline models. When we assessed the models for their capability to predict μ and τ, the epoch-versus-loss plot, as shown in Figure 3, revealed that the TraPPM model converges more rapidly compared to both NNLQP and MLP. This faster convergence can be attributed to TraPPM's ability to leverage unsupervised learning from unlabeled data.

To evaluate TraPPM's prediction capabilities in comparison to the baseline models, we used a test dataset and relied on the MAPE and RMSE metrics, as elaborated in Section A.3. Lower values of MAPE and RMSE are indicative of predictions being more accurate. A breakdown of the results for each model family is provided in Tables 3 and 4. The aggregate performance, presented in Table 1, highlights that TraPPM consistently delivers superior prediction accuracy for both μ and τ compared to the baselines. TraPPM achieved a relative improvement in MAPE of 40.6% over NNLQP, underscoring its robust performance in predicting μ. For predictions related to training step time, TraPPM consistently demonstrates superior accuracy when compared to other models. Notably, against NNLQP, TraPPM showed a relative improvement of 34.2% in MAPE, further emphasizing its proficiency in predicting step time.

Table 1: Average Performance Comparison of TraPPM with Baseline Models. The lower the value higher the accuracy.

	Memory Usage (MB)		**Step Time (ms)**	
Model	**MAPE**	**RMSE**	**MAPE**	**RMSE**
TraPPM	**4.92%**	**910.34**	**9.51%**	**23.23**
NNLQP	8.29%	1688.18	14.47%	37.02
MLP	85.01%	8045.68	134.07%	188.36
GBoost	16.10%	2971.52	16.98%	54.54

It is worth noting that the TraPPM model's semi-supervised approach, which leverages unsupervised training for generating embeddings, provides a competitive advantage in predicting μ and τ compared to the supervised training approach of NNLQP, which relies solely on labeled data. Overall, these results emphasize the efficacy of the TraPPM model, as it outperforms the majority of model families in predicting both μ and τ. We also explored the flexibility of TraPPM in predicting a new performance metric power consumption, represented as P_c. By utilizing the embeddings from the already trained GAE encoder, we simply retrained a GNN regressor tailored for P_c. Upon training for 100 epochs, TraPPM achieved a MAPE of 5.01% and an RMSE of 17 W. This underscores TraPPM's capacity to adeptly estimate diverse performance metrics with commendable accuracy. To provide visual evidence of TraPPM's effectiveness, we present the actual versus predicted plots in Figure 2. These plots exhibit a notable correlation between the predicted values by TraPPM and the actual outcomes, reflecting the TraPPM's superior prediction accuracy in forecasting training characteristics.

5 Conclusions

We present TraPPM, a novel framework that combines unsupervised GAE with a supervised GNN regressor to precisely predict DL model training characteristics without necessitating execution on target hardware. Our results highlight the potent advantages of a semi-supervised paradigm in DL resource prediction, consistently outstripping purely supervised approaches. TraPPM stands as a significant stride in DL performance prediction, facilitating better resource allocation. Complementing our methodology, we have made available an extensive dataset encompassing 25,053 unlabeled and 7,536 labeled DL graphs, fostering further research in this domain.

Acknowledgments and Disclosure of Funding

This work has been done in the context of the MAELSTROM project, which has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955513. The JU receives support from the European Union's Horizon 2020 research and innovation program and the United Kingdom, Germany, Italy, Switzerland, Norway, and in Luxembourg by the Luxembourg National Research Fund (FNR) under contract number 15092355.

References

Lu Bai, Weixing Ji, Qinyuan Li, Xilai Yao, Wei Xin, and Wanyi Zhu. Dnnabacus: Toward accurate computational cost prediction for deep neural networks, 2022.

Noureddine Bouhali, Hamza Ouarnoughi, Smail Niar, and Abdessamad Ait El Cadi. Execution time modeling for cnn inference on embedded gpus. In *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*, DroneSE and RAPIDO '21, page 59–65, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450389525.

Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. *NeuralPower*: Predict and deploy energy-efficient convolutional neural networks. In Min-Ling Zhang and Yung-Kyun Noh, editors, *Proceedings of the Ninth Asian Conference on Machine Learning*, volume 77 of *Proceedings of Machine Learning Research*, pages 622–637, Yonsei University, Seoul, Republic of Korea, 15–17 Nov 2017. PMLR.

Łukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. Brp-nas: Prediction-based nas using gcns. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Yanjie Gao, Yu Liu, Hongyu Zhang, Zhengxian Li, Yonghao Zhu, Haoxiang Lin, and Mao Yang. Estimating gpu memory consumption of deep learning models. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1342–1352, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370431.

Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. Runtime performance prediction for deep learning models with graph neural network. In *ICSE '23*. IEEE/ACM, May 2023. The 45th International Conference on Software Engineering, Software Engineering in Practice (SEIP) Track.

Eugenio Gianniti, Li Zhang, and Danilo Ardagna. Performance prediction of gpu-based deep learning applications. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 167–170, 2018. doi: 10.1109/CAHPC.2018.8645908.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 594–604, 2022.

Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the computational cost of deep learning models. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3873–3882, 2018.

Sam Kaufman, Phitchaya Phothilimthana, Yanqi Zhou, Charith Mendis, Sudip Roy, Amit Sabne, and Mike Burrows. A learned performance model for tensor processing units. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 387–400, 2021.

Liang Liu, Mingzhu Shen, Ruihao Gong, Fengwei Yu, and Hailong Yang. Nnlqp: A multi-platform neural network latency query and prediction system with an evolving database. In *51 International Conference on Parallel Processing - ICPP*, ICPP '22. Association for Computing Machinery, 2022.

Zongqing Lu, Swati Rallapalli, Kevin Chan, Shiliang Pu, and Thomas La Porta. Augur: Modeling the resource requirements of convnets on mobile devices. *IEEE Transactions on Mobile Computing*, 20(2):352–365, 2021.

Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Comput. Surv.*, 55(12), mar 2023. ISSN 0360-0300. doi: 10.1145/3578938.

Karthick Panner Selvam and Mats Brorsson. Dippm: A deep learning inference performance predictive model using graph neural networks, 2023.

Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *International Conference on Learning Representations*, 2017.

Max Sponner, Bernd Waschneck, and Akash Kumar. Ai-driven performance modeling for ai inference workloads. *Electronics*, 11(15), 2022. ISSN 2079-9292. doi: 10.3390/electronics11152316.

Delia Velasco-Montero, Jorge Fernández-Berni, Ricardo Carmona-Galán, and Ángel Rodríguez-Vázquez. Previous: A methodology for prediction of visual inference performance on iot devices. *IEEE Internet of Things Journal*, 7(10):9227–9240, 2020. doi: 10.1109/JIOT.2020.2981684.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC'21)*, 2021.

A Appendix

A.1 Enviroment setup

We used an AMD EPYC 7402 processor with two sockets (24 cores per socket), 512 GB DDR4-3200 RAM, and an NVIDIA A100 GPU with 40 GB HBM for data collection and experiment. The experimental environment for developing TraPPM involved the utilization of several essential Python libraries. The important libraries used were PyTorch 2.0.0, torch-geometric 2.3.0, torch-cluster, ONNX 1.13.1, and torch-sparse. These libraries played a crucial role in implementing and training the TraPPM model. The experiments for training TraPPM and generating the dataset were conducted on the abovementioned system using CUDA 11.7.

A.2 Datasets

For our TraPPM experiment, we employed a dual-method approach, harnessing both unsupervised and supervised datasets. As we already discussed, unsupervised datasets are used for training GAE, and the supervised dataset is used to train the GNN-based regressor. We utilized the Timm Wightman [2019] Python library to generate the unsupervised dataset for TraPPM, which offers various CNN and transformer-based architecture models. We saved these models in the ONNX format and converted them to PyG data, as described in Section 3. We created 25,053 DL models without labels, representing eleven different model families, as shown in Table 2. These models were utilized for unsupervised learning.

Our supervised dataset is a subset of the unsupervised dataset. The data collection was conducted on NVIDIA A100 GPU. We collected a total of 7536 labeled DL models. For baseline model comparisons, we utilized the labeled DL models. We again utilized the Timm library to generate DL models. However, instead of saving them to the ONNX format, we trained each model for 55 iterations, with the initial five iterations serving as a warm-up phase. We calculated the CUDA time during each iteration, representing the time taken to process a single iteration or step time in the training process. Our focus was primarily on step time, as it remains consistent during the training of the DL model, except for the initial few iterations that may exhibit variations due to warm-up effects. Therefore, we excluded the first five iterations when calculating the metrics. Additionally, we

Table 2: TraPPM: Dataset distribution

Family	Unsupervised	Supervised
densenet	838	466
efficientnet	1370	566
mnasnet	7208	795
mobilenet	2449	1613
poolformer	601	377
resnet	1805	821
swin	787	421
vgg	6171	937
visformer	237	235
convnext	1530	439
vit	2057	866
Total	**25053**	**7536**

collected memory usage and power consumption data using the NVML[1] Python library. For each DL model, we repeated this process, averaging the step time (ms), memory usage (MB), and power consumption (W). The results, along with the corresponding ONNX model files, were saved. While converting these models to PyG data format, we appended the measured values to the graph data.

A.3 Evaluation Metrics

To evaluate the performance of our TraPPM model relative to the baseline models, we leverage two primary evaluation metrics: Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE). MAPE, given by:

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|,$$

where y_i and $\hat{y}_i$ denote the actual and predicted values for the i^{th} observation respectively and n is the total number of observations, quantifies the average percentage deviation between predicted and actual values, offering a relative accuracy perspective. Conversely, RMSE, defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2},$$

assesses the aggregate magnitude of prediction errors, providing a standardized measure aligned with the scale of the predicted variable. By concurrently applying MAPE and RMSE, we furnish a comprehensive evaluation of TraPPM's predictive prowess against baseline models.

A.4 Baseline Models

In our evaluation, we compared TraPPM with three baseline models: Gboost, MLP, and the supervised GNN. Gboost served as a strong foundation for further development, while MLP was chosen for its wide usage in performance prediction Justus et al. [2018]. Finally, we included the supervised GNN model introduced by Liu et al. [2022], referred to as NNLQP. This model served as a reference for evaluating the performance of TraPPM in relation to a well-established supervised GNN approach.

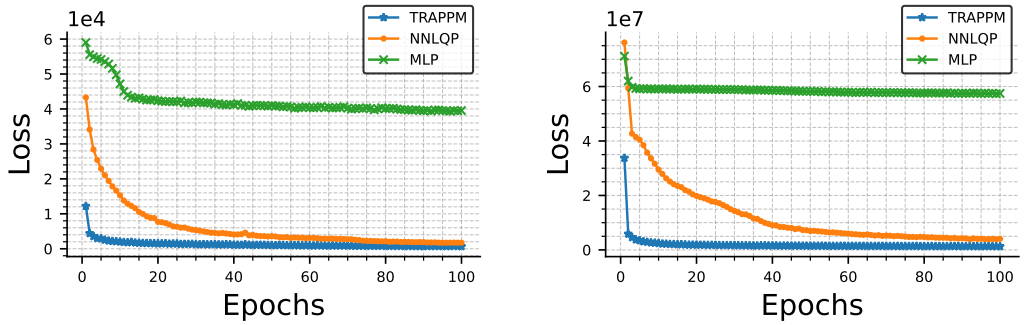
Gradient Boosting: To develop the GBoost model, we conducted training using the XGBoost python library. The training process involved utilizing a supervised dataset that solely consisted of DL static features as input. To optimize its hyperparameters, we conducted a grid search. The hyperparameters explored during the grid search were estimators with values [500, 1000, 2000], lr with values [1×10^{-3}, 1×10^{-4}], max depth with values [10, 30, 50], subsample with values

[1] https://pypi.org/project/pynvml/

[0.5, 0.75, 1], and colsample bytree with values [0.5, 0.75, 1]. After performing the grid search, we identified the best hyperparameters as follows: colsample bytree: 1, lr: 1×10^{-3}, max depth: 50, estimators: 2000, subsample: 1.

MLP: We created a baseline MLP model that is similar to the TraPPM MLP component, with the only difference being that it accepts only static features as input during training. We trained the baseline MLP model using 100 epochs, utilizing the MSE loss function and the Adam optimizer with lr=1×10^{-3}, which is the same setting as the TraPPM supervised training.

NNLQP: It is important to note that a key distinction between the NNLQP model and the TraPPM model is that the NNLQP model is unable to utilize unsupervised datasets. It can only operate with supervised datasets. To ensure a fair comparison, we kept the model architecture unchanged, only adapting the node features to accommodate the TraPPM dataset as discussed in Section 3. We trained the model for 100 epochs using the Adam optimizer with lr=1×10^{-3}, following the same settings as the TraPPM model. The NNLQP model takes the graph representation G as input, generates embeddings, concatenates them with static features, and employs an MLP to predict performance.



(a) Step Time (ms)

(b) Memory Usage (MB)

Figure 3: Epoch vs Loss plot comparing the convergence rates of TraPPM, NNLQP, and MLP. TraPPM showcases rapid convergence due to its ability to leverage unsupervised learning from unlabeled data, as trained over 100 epochs.

Table 3: Comparative Performance Analysis of Memory Usage (MB) Prediction: Averaged results over five distinct splits. Results highlight TraPPM's enhanced accuracy compared with baseline models. The lower the values, the higher the accuracy.

Family	**TraPPM**		**NNLQP**		**MLP**		**GBoost**	
	MAPE	**RMSE**	**MAPE**	**RMSE**	**MAPE**	**RMSE**	**MAPE**	**RMSE**
convnext	**4.95%**	**1005.71**	7.01%	1490.67	54.74%	8906.85	14.62%	3230.67
densenet	**3.52%**	**730.47**	8.29%	1675.17	66.44%	8317.23	14.64%	3113.40
efficientnet	**3.55%**	**537.84**	7.67%	1687.05	51.70%	11952.91	16.70%	3458.01
mnasnet	**4.53%**	**585.65**	6.75%	1804.18	94.42%	4908.33	14.72%	2756.27
mobilenet	**5.28%**	**633.74**	6.74%	1587.84	108.22%	5051.35	24.65%	2879.35
poolformer	**4.41%**	**1441.70**	6.96%	2027.24	76.10%	8951.67	15.04%	3332.57
resnet	**4.65%**	**658.40**	8.09%	1229.99	124.26%	7393.93	16.78%	2479.84
swin	**5.09%**	**774.91**	10.37%	1853.26	53.68%	8294.61	15.12%	2909.59
vgg	**10.48%**	2341.17	10.76%	**2271.89**	42.29%	7145.38	15.87%	3911.28
visformer	**3.92%**	**318.97**	9.49%	722.83	191.19%	8671.54	13.97%	1170.94
vit	**3.78%**	**985.22**	9.07%	2219.88	72.05%	8908.69	14.99%	3444.81

Table 4: Comparative Performance Analysis of Step time (ms) Prediction: Averaged results over five distinct splits. Results highlight TraPPM's enhanced accuracy compared with baseline models. The lower the values, the higher the accuracy.

Family	**TraPPM**		**NNLQP**		**MLP**		**GBoost**	
	MAPE	**RMSE**	**MAPE**	**RMSE**	**MAPE**	**RMSE**	**MAPE**	**RMSE**
convnext	**8.09%**	**46.00**	9.50%	61.06	64.92%	354.59	16.15%	102.72
densenet	**6.69%**	**15.46**	18.41%	36.50	104.45%	155.09	14.08%	33.61
efficientnet	**6.81%**	**13.46**	9.45%	22.51	49.18%	118.56	15.36%	34.94
mnasnet	**7.98%**	**12.72**	18.59%	40.05	106.86%	80.87	14.49%	41.18
mobilenet	**9.20%**	**8.95**	14.66%	21.69	116.48%	52.97	25.79%	31.44
poolformer	**13.02%**	27.05	13.23%	**26.79**	166.75%	119.56	14.59%	32.51
resnet	**11.26%**	**16.20**	25.45%	36.02	192.95%	122.75	24.13%	46.33
swin	9.01%	35.18	**8.89%**	**33.66**	60.08%	263.86	15.68%	72.44
vgg	**10.74%**	**22.51**	13.20%	30.29	69.91%	83.70	15.89%	43.59
visformer	14.79%	17.88	18.61%	15.29	437.33%	287.67	**13.99%**	**14.85**
vit	**7.06%**	**40.13**	9.15%	83.41	105.84%	432.32	16.60%	146.39