Can Tree-Based Model Improve Performance Prediction for LLMs?

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

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

***Abstract*—The deployment of Large Language Models (LLMs) in cloud environments underscores the imperative for optimal hardware configurations to enhance efficiency and reduce environmental impact. Prior research on performance prediction models predominantly focuses on computer vision, leaving a void for models adept at the unique demands of LLMs. This study bridges that gap, evaluating the potential of Tree-based models, particularly XGBoost, against traditional Graph Neural Networks (GNNs) in predicting the performance of LLMs. Our analysis shows that XGBoost achieves significant improvements in predicting throughput, memory usage, and energy consumption showcasing relative enhancements of MAPE approximately 68.81%, 80.85%, and 88.21%, respectively, compared to the GNN baseline, with a remarkable speed enhancement of approximately 26761.39% over GNN. These findings underscore XGBoost's effectiveness in accurately forecasting LLM performance metrics, offering a promising avenue for hardware configuration optimization in LLM deployment.**

***Index Terms*—Performance Model, Large Language Model, Graph Neural Network**

I. Introduction

The rapid evolution of Large Language Models (LLMs) has been a cornerstone of recent advancements in natural language processing, enabling unprecedented capabilities in text generation, comprehension, and translation. Models such as BERT, GPT, and their variants have demonstrated the profound impact of deep learning in understanding and generating human language. However, the deployment of these sophisticated models poses significant challenges, chiefly due to their substantial computational demands and the environmental impact of their operation. As LLMs become increasingly central to a wide range of applications, the imperative to optimize their deployment on various hardware configurations becomes evident, not only to enhance computational efficiency but also to mitigate costs and reduce carbon emissions.

LLM performance across different hardware platforms is influenced by a myriad of factors, including throughput (T), peak memory usage (M_{peak}), energy consumption per sample (E) as shown in Figure 1. These metrics are critical for selecting the optimal hardware configuration that balances performance with energy efficiency and environmental sustainability. Yet, accurately predicting these metrics presents a complex challenge, compounded by the intricate architecture

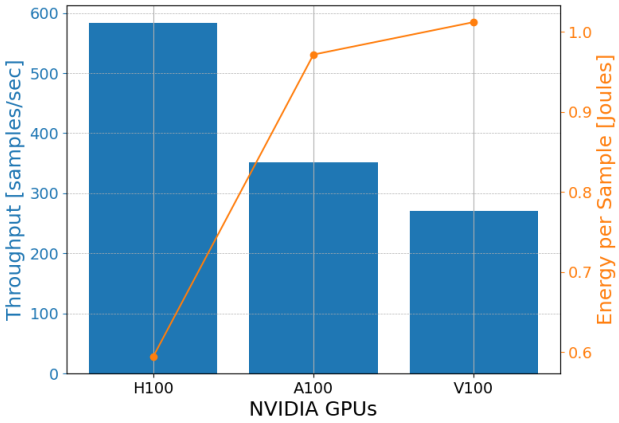


Fig. 1: Performance analysis of running DeBERTa V3 Large model with batch size 512 and sequence length 64 on Nvidia H100, A100, and V100 GPUs, comparing throughput (samples/sec) and energy per sample (J).

of LLMs and the dynamic nature of hardware utilization. Considering the importance of hardware selection in optimizing performance and reducing environmental impact, it is essential to thoroughly analyze these metrics when deploying LLMs.

Historically, the prediction of deep learning model performance has relied on Graph Neural Networks (GNNs), which, while effective in general contexts, face challenges when applied to LLMs. LLMs are characterized by their extensive node and edge counts, as demonstrated in Table I, and a repetitive layer structure that complicates traditional performance prediction methods. To illustrate, Figure 2 displays the frequency of operations within the BERT model, revealing a repetitive stacking of encoder layers. Each layer comprises distinct multi-head attention and feed-forward networks. This repetitive structure is not just a superficial characteristic but a crucial factor influencing performance across diverse hardware configurations.

Acknowledging these hurdles, our study advocates for a departure from graph-based representations to tabular formats when predicting LLM performance. We posit that this tran-

TABLE I: Total Nodes and Edges of LLMs

Model	Nodes	Edges
bert-large-uncased [4]	2896	6621
xlm-roberta-base [3]	1495	3417
roberta-large [16]	2923	6681
microsoft-deberta-v3-small [12]	2450	5379
roberta-base [16]	1495	3417
bert-base-uncased [4]	1468	3357
distilbert-base-uncased [19]	685	1579
microsoft-deberta-v3-large [12]	9398	20643
microsoft-deberta-v3-base [12]	4766	10467

sition could potentially improve both accuracy and prediction speed owing to the hierarchical and repetitive nature inherent in LLM architectures. To investigate this hypothesis, we curated a unique dataset comprising nine distinct LLMs by varying batch and sequence lengths and captured metrics across three NVIDIA GPUs: H100, A100, and V100. This dataset is augmented with comprehensive model and hardware features, enabling an exhaustive assessment of XGBoost against conventional GNN methodologies.

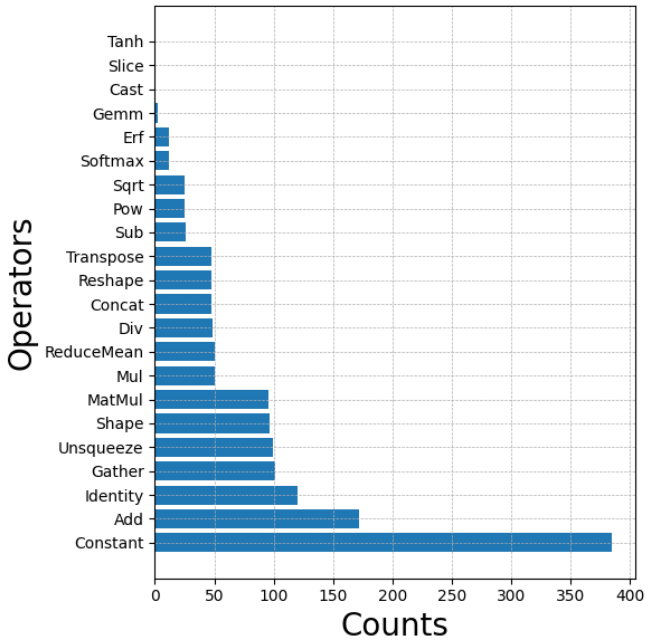


Fig. 2: Frequencies of different operations (ops) in the BERT base model. The figure illustrates the repetitive nature of certain operations within the model's architecture, highlighting the prevalence of operations like 'Constant', 'Identity', and 'Add'.

Furthermore, we have developed an intuitive user interface that simplifies the performance prediction process. Users can input the Hugging Face model ID, batch size, sequence length, and the number of forward passes, alongside their hardware choice, to receive predictions on throughput, peak memory, energy consumption, and carbon emissions within a few seconds. This tool is designed to empower users to make informed decisions about hardware configurations without the need for extensive computational resources.

Contributions:

- Intuitive Tree-based Model: Introducing XGBoost as a solution, we effectively address the complex hierarchical structures of LLMs, demonstrating superior performance in prediction tasks.
- Comprehensive Feature Selection Strategy: By meticulously selecting features that encapsulate the architecture of LLMs and the specifications of hardware, we facilitate nuanced predictions of model performance across various platforms.
- Creation of a Unique Dataset: This first-of-its-kind dataset, detailing LLM performance on NVIDIA's H100, A100, and V100 GPUs, serves as a vital resource for benchmarking and further research into hardware optimization for AI deployment.
- Through these contributions, our research not only tackles the practical challenges of LLM deployment but also fosters discussions on sustainable AI practices, paving the way for the development of more efficient and environmentally friendly AI systems.

TABLE II: Comparison with Recent Work on Performance Prediction. Our model is the first to predict the throughput (T), carbon emissions (C), and peak memory usage (M_{peak}) for LLMs.

Previous Work	LLM	C	T	(M_{peak})
[15] NNLQP (2022)	x	x	✓	x
[20] DIPPM (2023)	x	✓	✓	✓
[22] NarformerV2 (2023)	x	x	✓	x
[17] TraPPM (2023)	x	✓	✓	✓
[6] LLM Carbon (2024)	✓	✓	x	x
[11] CDMPP (2024)	(limited)	x	✓	x
Ours	✓	✓	✓	✓

II. Related Work

The field of performance prediction for deep learning (DL) models, especially in the context of hardware optimization, has witnessed growing interest in recent years. Early work by Qi et al. [18] proposed an analytical model to estimate DL model training times, but its accuracy was limited due to the neglect of concurrent operations. Subsequent studies like that of Gao et al. [8] extended these methods to predict memory consumption, utilizing analytical models to estimate resource utilization during training.

To improve prediction accuracy, researchers have explored machine learning approaches. Bouhali et al. [2] used an MLP-based regressor with features like trainable parameter counts, but it was constrained by a shallow understanding of DL layers' dynamics. Others, such as Justus et al. [13] and Gianti et al. [9], adopted a layer-by-layer technique, incorporating parameters like FLOPs to predict execution times and power consumption, showcasing progress towards more nuanced models.

In contrast, some researchers opted for a GNN over an MLP in a layerwise approach for DL model performance prediction [14], [5]. However, this layerwise strategy failed to capture the network structure of DL models [15]. To address this limitation, Gao et al. [7] and other researchers [15], [1], [20], [17], [22] utilized graph learning techniques to generate embeddings that encapsulate the model network topology. These embeddings were then combined with overall DL features to predict training and inference characteristics accurately.

Previous work in performance prediction has predominantly focused on computer vision tasks rather than LLMs. While GNNs have shown effectiveness in computer vision tasks, their performance can be hindered when applied to LLMs due to the larger number of nodes and edges in the graphs representing LLM architectures. GNNs typically involve message passing and node aggregation, which can lead to significant computational overhead and information loss when processing such large and complex graphs. Notably, extensive studies or datasets are scarce for LLMs, with most existing models tailored for computer vision tasks as shown in Table II. For instance, the CDMPP [11] works have created performance models for computer vision tasks, with only slight utilization of the BERT-base model [4]. This scarcity underscores the need for dedicated performance models specifically tailored for LLMs, prompting our novel approach leveraging Tree-based models, particularly XGBoost, to accurately predict LLM performance across diverse hardware configurations.

In our research, we establish a thorough dataset encompassing performance metrics for nine LLMs, systematically varying batch size and sequence length across three NVIDIA GPU architectures, as illustrated in Table I, and Table IV, respectively. Our work represents a groundbreaking initiative in the field, marking the inaugural development of a predictive performance model tailored specifically for LLM. This pioneering endeavor constitutes the first comprehensive study in LLM performance prediction.

III. Methodology

This section outlines our methodology for predicting key performance metrics of LLMs, focusing on T, M_{peak}, E, and carbon emissions. We begin by discussing the impact of parameters on LLM performance, followed by our approach to feature selection based on thorough analysis and data collection. Finally, we describe the evaluation metrics used to assess the performance of our predictive models.

T: Measured in samples per second, throughput is a critical metric for evaluating the processing speed and efficiency of LLMs. High throughput indicates a model's ability to quickly process data, enhancing user experience and operational productivity.

$$T = \frac{\text{Number of Samples Processed}}{\text{Time Taken (seconds)}} \tag{1}$$

$M_{\textbf{peak}}$: Represented in megabytes (MB), peak memory usage determines the maximum amount of memory required during model execution. This metric is vital for assessing whether a model can be deployed on specific hardware configurations without out-of-memory error.

E: Expressed in joules per sample, this metric provides insights into the energy efficiency of running LLMs, directly correlating with operational costs and environmental footprint.

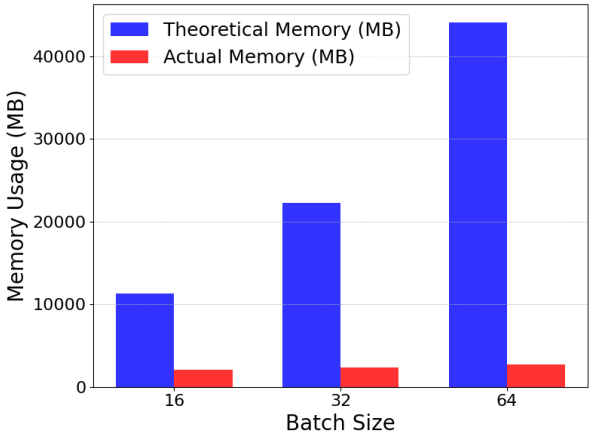


Fig. 3: Comparison of Theoretical and Actual Memory Usage for BERT-base-uncased with batch size 256 on the H100 device. Theoretical memory calculations were obtained using the ONNX tool.

A. Empirical Analysis of Memory Usage

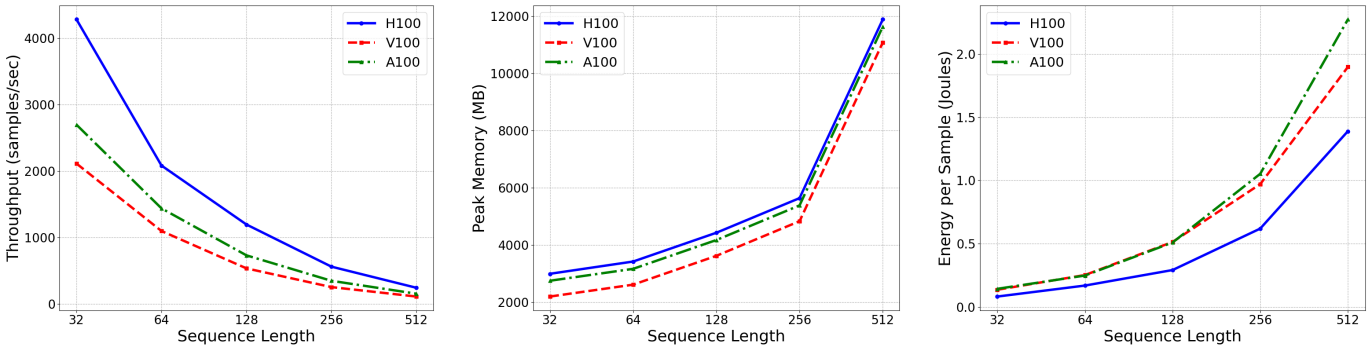
We address the discrepancies between theoretical and actual memory requirements through empirical analysis, as shown in Figure 3. Theoretical estimates often overlook runtime overheads and dynamic memory allocation strategies employed by deep learning frameworks, leading to inaccurate hardware selection. This necessitates an empirical approach to performance prediction.

B. Carbon Emission Calculation

Carbon emissions (C) result from the energy consumed per sample (E) multiplied by the carbon intensity of the electricity source (CI), with the total carbon emissions given by $C = E \times CI$. This calculation is essential for evaluating the environmental impact of deploying LLMs and is influenced by factors such as the number of samples processed.

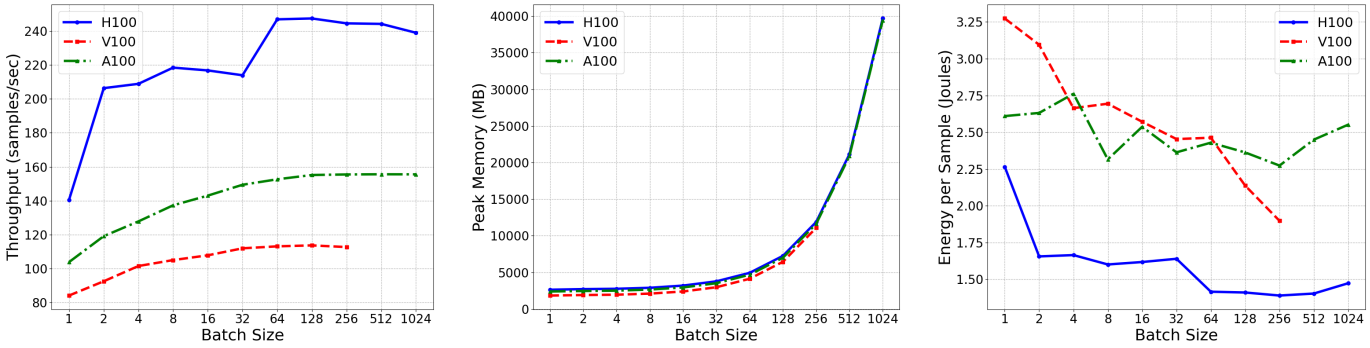
C. Influence of Batch Size and Sequence Length

Our investigation, depicted in Figures 4 and 5, illuminates the impact of sequence length and batch size on LLM performance metrics (T, M_{peak}, E) across diverse Nvidia hardware configurations (H100, A100, and V100). Notably, we observe a substantial decrease in T as sequence length increases, reflecting the computational complexity inherent in processing longer sequences. However, this trend is offset by an increase in T with larger batch sizes, showcasing the intricate relationship between batch processing and computational efficiency.



(a) Throughput vs. Sequence Length (b) Peak Memory vs. Sequence Length (c) Energy per sample vs. Sequence Length

Fig. 4: Comparative analysis of throughput, peak memory, and energy per sample for varying sequence lengths of Roberta XLM model with a fixed batch size of 256.



(a) Throughput vs. Batch Size (b) Peak Memory vs. Batch Size (c) Energy per sample vs. Batch Size

Fig. 5: Comparative analysis of throughput, peak memory, and energy per sample for varying batch sizes of Roberta XLM model with a fixed sequence length of 512.

Furthermore, our analysis unveils the influence of sequence length and batch size on M_{peak}, E. As sequence length increases, M_{peak} escalates due to the greater memory demands of processing longer input sequences, a trend consistent across all hardware configurations. Conversely, the impact of batch size on M_{peak} is akin to that of varying sequence length because increasing batching allocates more High Bandwidth Memory (HBM), facilitating efficient memory utilization. However, increasing batch size results in a reduction in energy consumption per sample, as the hardware can effectively handle larger batch sizes, optimizing energy efficiency during processing.

D. Quantization Effects on LLM Performance

Quantization strategies, such as FP32 (Single Precision Floating Point), FP16 (Half Precision Floating Point), and BNB 4bit (BitsAndBytes[1]), significantly impact the performance metrics of LLMs. Our analysis of the LLaMA 7B model [21] on the A100 device reveals distinct outcomes for each quantization strategy. Transitioning from FP32 to FP16 quantization results in a substantial reduction in M_{peak}, as shown in Figure 6. This reduction indicates that FP16 representation requires less memory for model storage and com-

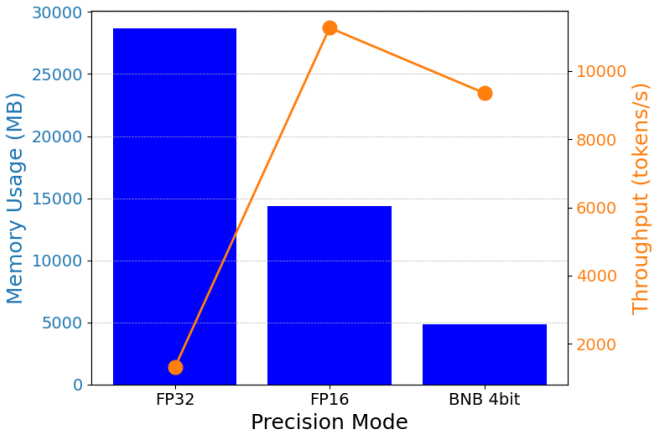


Fig. 6: Comparison of Quantization Strategies (FP32, FP16, and BNB 4bit) for the LLaMA 7B model on the A100 device. The figure illustrates the impact of quantization on peak memory usage and throughput.

[1]https://github.com/TimDettmers/bitsandbytes

putation. Consequently, the throughput increases significantly from FP32 to FP16. Adopting BNB 4bit quantization further reduces memory usage to 4848.55 MB while maintaining a high throughput. The E decreases from 2.6×10^{-5} J (FP32) to 1.6×10^{-5} J (FP16) and then increases slightly to 1.8×10^{-5} J for BNB 4bit quantization, showcasing the trade-off between M_{peak} usage and E.

TABLE III: Model and Hardware Features

Feature (Abbreviation)	Description
Model Features (MF)	
BS	Batch Size
SL	Sequence Length
QT	Quantization Type
TN	Total Nodes
TE	Total Edges
TM	Total MACs
TP	Total Parameters
TM_{mem}	Total Memory
MPE	Max Position Embeddings
NHL	Num Hidden Layers
NAH	Num Attention Heads
HS	Hidden Size
VS	Vocab Size
TVS	Type Vocab Size
Hardware Features (HF)	
PF_{32}	Peak FP32 GFLOPS
PB_{HBM}	Peak Bandwidth HBM
CM	Compute Major
Cm	Compute Minor
MTPB	Max Threads Per Block
MTPM	Max Threads Per Multiprocessor
RPB	Regs Per Block
RPM	Regs Per Multiprocessor
WS	Warp Size
SMPB	Shared Mem Per Block
SMMP	Shared Mem Per Multiprocessor
SMS	Num Streaming Multiprocessors
SMPBO	Shared Mem Per Block Opt-In
CC	CPU Clock
RAM	System RAM
LC1	L1 Cache
LC2	L2 Cache
LC3	L3 Cache
NC	Num Cores
CA	CPU Architecture

E. Feature Selection for Performance Prediction

To predict LLM performance metrics (T), (M_{peak}), and (E), we've compiled a comprehensive set of model and hardware features in the Table III.

We measured the computational capabilities and memory bandwidth of Nvidia GPUs using two methods. We ran a CUDA[2] kernel for intensive floating-point calculations for (PF_{32}). Another CUDA kernel measured the data transfer rate for memory bandwidth, yielding peak bandwidth (PB_HBM) in GB/s. The other hardware features are directly extracted from CUDA APIs.

Model features were extracted by converting the LLMs to the ONNX format, facilitating the use of ONNX tool[3] for calculating (TM_{mem}), (TM), and (TP). Additionally, we traversed the ONNX graph to calculate the (TN) and (TE), providing a comprehensive understanding of the model's computational graph. The other model metrics like (HS), (NHL), (NAH), (VS), and (TVS) were directly extracted from the Hugging Face library[4] using the model ID.

Algorithm 1 LLM Performance Metrics Collection

Require: $B = \{b_1, b_2, \ldots, b_n\}$, $S = \{s_1, s_2, \ldots, s_m\}$, $M = \{m_1, m_2, \ldots, m_k\}$, H, W, T
Ensure: Dataset $\mathcal{D}$

```
for p ∈ P do
    Initialize model m, tokenizer τ
    for b ∈ B do
        for s ∈ S do
            if config (m, b, s) not in D then
                CoolDownGPU
                Initialize trackers
                Warm up M for W iterations using τ
                Record metrics for T iterations
                Calculate T, M_peak, E_sample, and C
                D ← D ∪ {(m, b, s, T, M_peak, E, C)}
            end if
        end for
    end for
end for
function CoolDownGPU
    Clear CUDA cache and collect garbage
    Sleep for cooldown period
end function
```

IV. Dataset Collection

Our dataset collection methodology enables a comprehensive analysis of LLMs across diverse computational paradigms, focusing on tree-based models and GNN baselines. To gather data, we conducted forward passes across various LLMs and hardware configurations to empirically measure T, M_{peak}, and E. This process involved initializing models and tokenizers, configuring CUDA settings, and systematically iterating over batch sizes B and sequence lengths S. Utilizing PyTorch for model operations and specialized trackers for latency, memory, and energy metrics, we ensured accurate measurements with a cooldown phase between iterations. We employed 100 warmup iterations W and 300 total iterations T for data collection.

The collected metrics were then compiled into a dataset $\mathcal{D}$, divided into tabular and graph formats for distinct analytical approaches. Algorithm 1, outlines the process for collecting LLM performance metrics, taking inputs such as batch sizes B, sequence lengths S, LLM models P, hardware configuration H, warmup steps W, and total steps T, and producing the dataset $\mathcal{D}$ containing the measured metrics.

[2]https://developer.nvidia.com/cuda-toolkit

[3]https://github.com/ThanatosShinji/onnx-tool

[4]https://huggingface.co/

TABLE IV: Hardware Specifications of Data Collection Devices

Specification	Juwels	Booster	Jueraca
CPU	2× Intel Xeon Gold 6148	2× AMD EPYC Rome 7402	2× Intel Xeon Platinum 8452Y
Cores	2× 20 cores, 2.4 GHz	2× 24 cores, 2.8 GHz	2× 36 cores, SMT-2
Memory	192 GB DDR4, 2666MHz	512 GB DDR4, 3200 MHz	512 GiB DDR5-4800 RAM
GPU	NVIDIA V100, 16 GB HBM	NVIDIA A100, 40 GB HBM2e	NVIDIA H100, 80 GB HBM
Network	2× InfiniBand EDR	4× InfiniBand HDR	1× BlueField-2 ConnectX-6

A. Tabular and Graph Dataset Construction

For tree-based models, we organize performance metrics and extracted features into tabular format, saved as CSV files. This dataset directly aids algorithms like XGBoost, facilitating efficient performance prediction of LLMs across hardware setups.

To exploit relational information for GNN baseline comparison, we convert LLMs (in ONNX[5] format) into graphs compatible with PyTorch Geometric (PyG[6]). Each DL model, denoted as M with operations O, transforms into a graph $G = (X, A)$, where X and A represent node feature and adjacency matrices. Nodes in G correspond to operations in M, capturing computational characteristics and information flow. This transformation encodes operation types, input/output shapes, MACs, parameters, and memory requirements into node attribute vectors, enhancing graph representation fidelity. We utilized the same methodologies as previously published work for constructing graphs [17].

B. Evaluation Metrics

We assess model performance using three key metrics: Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and Kendall's Tau (τ). RMSE measures the average deviation between predicted and actual values, while MAPE quantifies the percentage error. Kendall's Tau evaluates the correlation between predicted and actual rankings.

V. Experiments and Results

A. Enviroment setup

We utilized three distinct hardware configurations for data collection, outlined in Table IV. Our experiments were specifically conducted on systems featuring NVIDIA A100 GPU. Additionally, we employed PyTorch 2.2.1, torch-geometric 2.5.0, ONNX 1.13.1, CUDA 12.1, and XGBoost 2.0.3 for the experimental setup.

B. GNN Baseline Model

Our GNN baseline model architecture takes an input graph $G = (X, A)$, where X captures node features and A represents the adjacency matrix. Two GraphSAGE [10] (GS) layers transform the node features: $X' = \text{ReLU}(\text{Dropout}(GS(X, A; \theta_1)), p = 0.05)$ and $X'' = \text{ReLU}(\text{Dropout}(GS(X', A; \theta_2)), p = 0.05)$. These layers extract meaningful representations from the input features. Static features S contains static $= [TM, TP, TM_{\text{mem}}, BS, SL, HF]$ undergo linear transformation to align their dimensionality with the output of the GS layers: $S' = \text{ReLU}(\text{Dropout}(\text{FC}_{S \to H_{\text{gnn}}}(S)), p = 0.05)$. The processed static features S' are concatenated with X'' to create a unified feature representation. The concatenated features are fed into fully connected layers, resulting in the final prediction y: $y = \text{FC}_{H_{\text{fc}} \to 1}(F')$, where $H_{\text{fc}} = 512$.

The model minimizes L_{MSE} using Adam (lr=0.001) over mini-batches of size 16. The choice of the GNN architecture is motivated by its widespread use in previous literature [15], although previous implementations often lacked hardware-related properties in their static features. We incorporated these hardware-related properties into the static features to ensure a fair comparison with the XGBoost model. This integration enables the Baseline model to effectively capture complex relationships among input graphs, hardware configurations, and model features, facilitating a more robust comparison with the XGBoost model.

C. Our Model: XGBoost

Our approach to predicting LLM performance metrics utilizes the XGBoost algorithm, renowned for its efficiency with structured data. XGBoost optimizes an objective function Obj, composed of a loss function L and a regularization term Ω, defined as:

$$\text{Obj} = \sum_i L(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{2}$$

where y_i represents true metric values, $\hat{y}_i$ denotes predictions, and f_k are decision trees. The loss function L measures prediction discrepancy, while Ω penalizes model complexity to prevent overfitting.

Hyperparameter tuning involves a grid search over parameters such as the number of estimators: [100, 500, 1000, 2000], learning rate: [1×10^{-1}, 1×10^{-2}, 1×10^{-3}], maximum tree depth: [3, 5, 8, 10, 30], and subsample ratios: [0.5, 0.75, 1]. The optimal set of hyperparameters includes a learning rate of 1×10^{-1}, a maximum depth of 8, 2000 estimators, and a subsample ratio of 1.

VI. Experimental Evaluation

In our experimental evaluation, we aimed to assess the predictive capabilities of GNN and XGBoost models regarding T, M_{peak}, and E for LLMs. This assessment was conducted within the framework of stratified 5-fold cross-validation to ensure thorough and unbiased evaluation.

Each model, denoted by $\mathcal{M} \in \{\text{GNN}, \text{XGBoost}\}$, was trained on the training set and evaluated on the test set across

[5]https://onnx.ai/

[6]https://www.pyg.org/

TABLE V: Comparison of GNN and XGBoost Models Across Different Metrics

Target	GNN			XGBoost		
	MAPE (%) ↓	RMSE ↓	τ ↑	MAPE (%) ↓	RMSE ↓	τ ↑
Throughput	17.60	481.82	0.852	5.49	109.73	0.96
Memory	9.45	2657.45	0.874	1.81	1209.22	0.98
Energy	53.18	1.94	0.41	6.27	0.20	0.96

5 folds. The predictions for T, M_{peak}, and E for each fold are then averaged to quantify the overall performance of the model.

A. Baseline Comparison

The comparison between GNN and XGBoost models reveals stark differences in predictive accuracy, as shown in Table V. XGBoost's superior performance may stem from its adeptness at handling structured data and leveraging ensemble learning techniques. In contrast, GNNs may struggle to capture complex relationships in LLM data due to inherent limitations in modeling graph structures effectively.

The observed disparities may also be attributed to the algorithmic complexity of each model. XGBoost optimizes for predictive accuracy through iterative improvement, while GNNs rely on message-passing mechanisms, potentially hindering their ability to capture nuanced patterns in LLM data.

B. Ablation Study

We conduct an ablation study to assess the impact of feature inclusion and hardware specifics on the XGBoost model as shown in Table VI. The study examines different feature sets: MAC operations TM, combined MACs with parameters and memory $TM+TP+TM_{\text{mem}}$, and all available model features MF, across three key metrics: T, M_{peak}, and E.

Initially, TM features provide baseline accuracy, with notable enhancements observed when incorporating model parameters and memory usage $TM + TP + TM_{\text{mem}}$. Integrating hardware details, including FLOPS and bandwidth, significantly boosts predictive performance. Utilizing the full model feature set alongside hardware features yields the most substantial improvements, emphasizing the pivotal role of hardware in computational performance.

C. Efficiency Analysis of Predictive Models

Our evaluation rigorously compares the computational efficiency of the Baseline GNN and XGBoost models, focusing on throughput as a crucial performance metric. The experiments were conducted on Booster CPUs specified in Table IV. As depicted in Figure 7, XGBoost exhibits markedly superior efficiency in predicting performance for LLMs. This efficiency advantage is attributed to XGBoost's optimized algorithms, specifically tailored for handling structured data and large datasets. In contrast, GNNs, encounter challenges due to the intricate graph structures inherent in LLMs.

D. Usability

We developed a user-friendly interface developed with Gradio[7], streamlining the prediction of performance metrics (T), M_{peak}), and carbon emissions (C) for LLMs. This interface allows users to input a model ID from the Hugging Face repository, batch size, sequence length, and device selection (H100, A100, V100), or directly input hardware features. Behind this interface lies a backend that transforms user inputs into a feature set as mentioned in Table III, then features are passed to pre-trained XGBoost models to predict performance metrics accurately. This seamless integration of a user-friendly interface with powerful predictive capabilities signifies a significant advancement in optimizing LLM deployments, fostering more efficient and environmentally conscious AI applications.

[7]https://www.gradio.app/

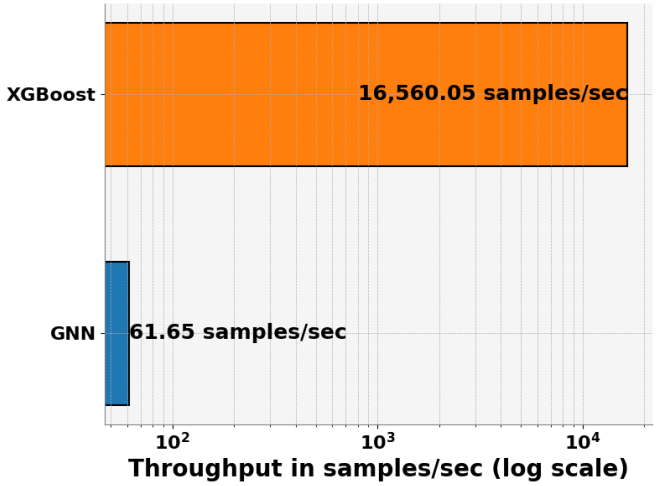


Fig. 7: Comparison of model throughput on CPU between GNN and XGBoost models, illustrating the significant efficiency advantage of the XGBoost model in processing samples per second.

VII. Conclusion

Our study demonstrates the superiority of XGBoost over Baseline GNN in predicting Large Language Model (LLM) performance across diverse hardware configurations. XGBoost achieves substantial improvements in throughput, memory usage, and energy consumption prediction accuracy, with relative enhancements of MAPE approximately 68.81%, 80.85%, and 88.21% compared to the GNN baseline, respectively. Remarkably, XGBoost achieves a remarkable 26761.39% relative increase in throughput speed over GNNs.

These findings highlight XGBoost's efficiency in rapid prediction processing, providing a strong basis for optimizing hardware selection strategies. By enhancing computational efficiency and advocating for sustainable LLM deployment,

TABLE VI: Ablation Study on the Effect of Model (MF) and Hardware Features (HF) on Prediction Metrics (Throughput (T), Peak Memory Usage (M_{peak}), and Energy Consumption (E))

Metrics	Features	Without HF			With HF		
		MAPE (%) ↓	RMSE ↓	τ ↑	MAPE(%) ↓	RMSE ↓	τ ↑
T	TM	39.36	682.13	0.782	17.89	406.54	0.889
	$TM + TP + TM_{\text{mem}}$	43.25	695.95	0.764	6.16	116.17	0.959
	MF	43.19	695.79	0.764	5.49	109.73	0.962
M_{peak}	TM	24.21	1742.64	0.607	11.68	1785.61	0.804
	$TM + TP + TM_{\text{mem}}$	25.57	1660.02	0.608	1.94	1484.39	0.980
	MF	25.44	1468.69	0.608	1.81	1209.22	0.980
E	TM	31.08	0.77	0.833	14.82	0.42	0.912
	$TM + TP + TM_{\text{mem}}$	32.21	0.87	0.822	6.28	0.21	0.959
	MF	32.11	0.87	0.822	6.27	0.20	0.959

our research makes a significant contribution to the AI community, facilitating informed decision-making and promoting environmental consciousness in LLM deployment.

REFERENCES

[1] L. Bai, W. Ji, Q. Li, X. Yao, W. Xin, and W. Zhu, "Dnnabacus: Toward accurate computational cost prediction for deep neural networks," 2022.

[2] N. Bouhali, H. Ouarnoughi, S. Niar, and A. A. 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*, ser. DroneSE and RAPIDO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 59–65.

[3] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," 2020.

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

[5] L. Dudziak, T. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "Brp-nas: Prediction-based nas using gcns," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[6] A. Faiz, S. Kaneda, R. Wang, R. Osi, P. Sharma, F. Chen, and L. Jiang, "Llmcarbon: Modeling the end-to-end carbon footprint of large language models," 2024.

[7] Y. Gao, X. Gu, H. Zhang, H. Lin, and M. 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.

[8] Y. Gao, Y. Liu, H. Zhang, Z. Li, Y. Zhu, H. Lin, and M. 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*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1342–1352. [Online]. Available: https://doi-org.proxy.bnl.lu/10.1145/3368089.3417050

[9] E. Gianniti, L. Zhang, and D. Ardagna, "Performance prediction of gpu-based deep learning applications," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018, pp. 167–170.

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

[11] H. Hanpeng, S. Junwei, Z. Juntao, P. Yanghua, Z. Yibo, L. Haibin, and W. Chuan, "CDMPP: A Device-Model Agnostic Framework for Latency Prediction of Tensor Programs," in *Proceedings of the Nineteenth EuroSys Conference*, 2024.

[12] P. He, J. Gao, and W. Chen, "Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing," 2023.

[13] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 3873–3882.

[14] S. Kaufman, P. Phothilimthana, Y. Zhou, C. Mendis, S. Roy, A. Sabne, and M. Burrows, "A learned performance model for tensor processing units," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 387–400.

[15] L. Liu, M. Shen, R. Gong, F. Yu, and H. Yang, "Nnlqp: A multi-platform neural network latency query and prediction system with an evolving database," in *51 International Conference on Parallel Processing - ICPP*, ser. ICPP '22. Association for Computing Machinery, 2022.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019.

[17] K. Panner Selvam and M. Brorsson, "Can semi-supervised learning improve prediction of deep learning model resource consumption?" in *Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA*. [Online]. Available: https://openreview.net/forum?id=C4nDgK47OJ

[18] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *International Conference on Learning Representations*, 2017.

[19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[20] K. P. Selvam and M. Brorsson, "Dippm: a deep learning inference performance predictive model using graph neural networks," 2023.

[21] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[22] Y. Yi, H. Zhang, R. Xiao, N. Wang, and X. Wang, "Nar-former v2: Rethinking transformer for universal neural network representation learning," 2023.