Relative Performance Prediction using Few-Shot Learning

Arunavo Dey, Aakash Dhakal, Tanzima Z. Islam
Texas State University
{hcs77,nvc22,tanzima}@txstate.edu

Jae-Seung Yeom, Tapasya Patki
Lawrence Livermore National Laboratory
{yeom2,patki1}@llnl.gov

Daniel Nichols, Alexander Movsesyan, Abhinav Bhatele
University of Maryland
{dnicho,amovsesy,bhatele}@umd.edu

***Abstract*—High-performance computing system architectures are evolving rapidly, making exhaustive data collection for each architecture to build predictive performance models increasingly impractical. Concurrently, the arrival of new applications daily necessitates efficient performance prediction methods. Traditional data collection can take days or weeks, making it more efficient for scientists to leverage existing models to predict an application's performance on new architectures or use data from one application to predict another on the same architecture. The growing heterogeneity in applications and resources further complicates the exact matches needed for effective knowledge transfer. This work systematically studies various Machine Learning (ML) models to predict the relative performance of new applications on new platforms using existing data. Our findings demonstrate that few-shot learning using a few samples significantly enhances cross-platform knowledge transfer, multi-source models outperform single-source models, and Large Language Models (LLMs)-generated samples can effectively improve knowledge transfer efficacy.**

***Index Terms*—Performance Modeling, Machine Learning, Few-shot Learning, Large language models (LLM), Cross-platform performance prediction**

I. Introduction

Traditionally, performance modeling of an High Performance Computing (HPC) application is conducted in a data-driven manner by collecting many training samples from applications running on multiple platforms. These models aid in understanding application behaviors and predicting completion times, crucial for effective resource allocation in critical simulations like vaccine development. However, this method is platform-dependent and time-consuming, especially with the arrival of a new platform every six months. To avoid spending a long time on performance data collection, scientists often predict application performance on new platforms instead of data collection, saving time and resources.

However, the absolute performance of an application depends on interactions with the OS, software stack, and hardware. Existing supervised machine learning approaches [12, 14] require extensive data collection. We propose a comparative approach using machine learning to predict relative performance, avoiding the need to measure common factors like runtime and OS. Measurements include application parameters, machine characteristics, and dynamic interactions. We define the relative performance of an application $\mathcal{X}$ on platform $\mathcal{B}$ compared to platform $\mathcal{A}$ as $X_{B/A}$.

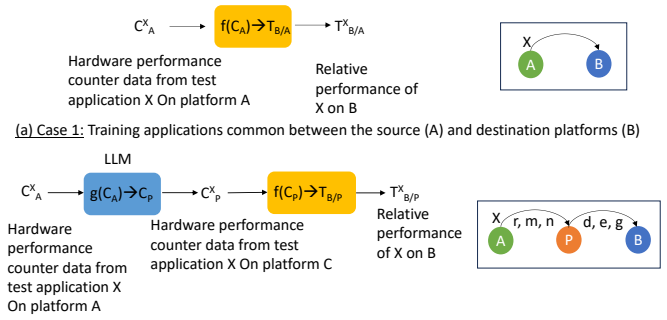
This study systematically examines various ML techniques for predicting $X_{B/A}$. We assess the effectiveness of state-of-the-art methods in two scenarios: (1) when at least one application is profiled on all available platforms, facilitating significant data collection efforts, and (2) when no training application is profiled on all platforms, but each pair has at least one profiled on both. Predicting relative performance based on data from other applications resembles visual categorization tasks such as object recognition and image classification [13]. If the source model can adapt using a few samples from the test application during transfer learning, it is called a few-shot learning approach [16]. While most literature focuses on zero-shot learning, we hypothesize that adjusting the source model with a few samples using few-shot learning can can enhance the model's generalizability.

Moreover, when training samples are scarce, we explore the applicability of the LLMs to generate performance samples to fill the gap to enable direct knowledge transfer across applications. Specifically, in this paper, we investigate the following research questions: (1) study the efficacy of several regression-based ML techniques when a relatively large number of labeled training samples are present in transferring knowledge; (2) whether a source model built using multiple applications' data is more effective than a single application's; (3) study whether leveraging a few new samples from the target application helps to improve ML model's efficacy further, and (4) in the presence of data scarcity, if a Large Language Model (LLM) can be used to generate data to alleviate the need for further data collection. We measure the efficacy of an ML model using the widely used Mean Square Error (MSE) metric, which calculates how different predictions are from the real values. $MSE = 0$ indicates a perfect prediction.

To summarize our contributions in this paper, we:

- Evaluate various ML approaches in predicting relative performance across platforms with sufficient training samples.
- Propose using a multi-source modeling approach and few-shot learning to enhance relative performance prediction efficacy.
- Introduce a novel predictive modeling method using LLMs to address data scarcity challenges.

Our unique HPC performance dataset comprises 3494 samples from running 6 applications (`Laghos`, `Kripke`, `SW4lite`, `TESTDFFT`, `miniVite`, `AMG`) on three platforms. Extensive evaluations with this dataset indicate that:



(a) Case 1: Training applications common between the source (A) and destination platforms (B)

(b) Case 2: No training application common between the source (A) and destination platforms (B). Assumption:

Fig. 1: Overview of the two scenarios that our proposed approach addresses.

(1) gradient-based modeling techniques perform well due to consistent data distribution across applications when sufficient training samples are present; (2) knowledge transfer from multiple-application source models is more effective than single-application models; (3) few-shot learning enhances relative performance prediction; and (4) leveraging generative AI, especially LLMs, to synthesize performance samples is effective when limited training data is present.

II. Methodology

A. Problem Formulation

The problem of cross-platform relative performance prediction can be formulated as a knowledge transfer problem in ML, where a source model trained on measurements from various applications collected on platforms $\mathcal{A}$ and $\mathcal{B}$ is used. The source model predicts the performance of an application $\mathcal{X}$ on platform $\mathcal{B}$ using its hardware performance counters on platform $\mathcal{A}$. Predicting relative instead of absolute performance enables avoidance of common factors such as runtime, OS, and software stack, which would otherwise necessitate an excessive number of performance samples.

Assuming performance samples with labels are available for a set of applications on platforms $\mathcal{A}$ and $\mathcal{B}$, measurements can encompass various factors like application details, machine specifications, and dynamic interactions with the system. For this study, hardware performance counters serve as features. In a typical scenario, a source model built from these measurements can predict an application's performance on $\mathcal{B}$ using its hardware counters on $\mathcal{A}$. However, in cases of data scarcity where no application was measured on both platforms, such a direct source model cannot be constructed.

Mathematically, these two scenarios can be summarized as:

- Case 1: Measurements from at least one application are available on both platforms, $\mathcal{A}$ and $\mathcal{B}$. ($App_A \cap App_B \neq \varnothing$).
- Case 2: No measurements from common applications are available on both platforms, $\mathcal{A}$ and $\mathcal{B}$. ($App_A \cap App_B = \varnothing$).

B. Case 1: Measurements from at least one application are available on both platforms, $\mathcal{A}$ and $\mathcal{B}$

Figure 1a shows what happens in Case 1. In Case 1, let's assume we have the hardware performance counter measurements from n common applications, $x_1, x_2, x_3, ..., x_n$ on platform $\mathcal{A}$, denoted by $C_A^{[x_1,x_2,x_3,...,x_n]}$. We also have the execution times for these applications available on platforms $\mathcal{A}$ and $\mathcal{B}$, denoted by $T_A^{[x_1,x_2,x_3,...,x_n]}$ and $T_B^{[x_1,x_2,x_3,...,x_n]}$. We calculate the ground truth values of their relative performance on $\mathcal{B}$ relative to $\mathcal{A}$ as $T_{B/A}^{[x_1,x_2,x_3,...,x_n]} = \frac{T_B^{[x_1,x_2,x_3,...,x_n]}}{T_A^{[x_1,x_2,x_3,...,x_n]}}$. In this case, we can leverage supervised learning as well as transfer learning approaches to predict the relative performance of a test application, $X_{B/A}$ using its hardware performance counter measurements, C_A^X, on $\mathcal{A}$.

C. Case 2: No measurements from common applications are available on both platforms, $\mathcal{A}$ and $\mathcal{B}$

Case 2 (Figure 1b) is a more challenging problem since building a source model to transfer knowledge directly between two platforms of interest is not viable due to the unavailability of a common application. **To simplify this scenario within the scope of this work, we assume that at least one application exists for each pair of platforms whose hardware performance counters are measured on both platforms.** That is, as shown in Figure 1b, the shortest path from $\mathcal{A}$ to $\mathcal{B}$ will contain at least one other platform $\mathcal{P}$. Let's assume that x_r, x_m, x_n are the applications with their data available on both $\mathcal{A}$ and $\mathcal{P}$. Whereas x_d, x_e, x_g are the applications with their data available on both $\mathcal{P}$ and $\mathcal{B}$.

To address the challenge of transferring knowledge across platforms where a direct knowledge transfer is not viable, we proposed a two-step solution.

Step 1: Data generation: We propose to leverage the tabular data generation capabilities of the LLMs to synthesize performance samples for test applications. During the training phase, we fine-tune an LLM with $C_A^{[x_r,x_m,x_n]}$ and $C_P^{[x_r,x_m,x_n]}$. This model learns to leverage one application's features on one platform to generate the features for the same application on another platform. During the test time, we leverage the fine-tuned LLM to generate performance samples for the test application $\mathcal{X}$ including hardware performance counters C_P^X and the runtime T_P^X on the platform $\mathcal{P}$.

Step 2: Building a predictive source model During the training phase, we build a source model similar to Case 1 with x_d, x_e, x_g applications' hardware performance counter data on platform $\mathcal{P}$, $C_P^{[x_d,x_e,x_g]}$ as features and their ground truth values of relative performances on $\mathcal{B}$ relative to $\mathcal{P}$, $T_{B/P}^{[x_d,x_e,x_g]}$ as target labels. During the testing phase, we leverage the synthesized hardware performance counter values as input to this source model to get the final predictions $T_{B/P}^X$.

III. Experimental Setup

A. Dataset

We leverage six different HPC proxy applications on three CPU and GPU platforms. Table I and [10] describe the applications `Laghos`, `Kripke`, `miniVite`, `AMG`, `SW4lite`, and `TESTDFFT` and the data collection process in detail. Table II describes the set of hardware performance counters used as features. Table III describes the specifications of the HPC platforms where data is collected.

B. Case Studies

In this work, we predict the relative performance of various applications on `Ruby` and `Corona` based on the hardware performance counter data of that application collected on

TABLE I: Description of the applications

Application	Description
Laghos	FEM for compressible gas dynamics
Kripke	3D Deterministic Particle Transport Code
miniVite	Graph Community Detection
TESTDFFT	Parallel 3D FFT
SW4lite	Seismic Wave Simulation
AMG	Algebraic Multi Grid Solver

TABLE II: Collected hardware performance counters

Feature	Description
Branch Intensity	Ratio of branch instructions to total instructions
Load Intensity	Ratio of load instructions to total instructions
Store Intensity	Ratio of store instructions to total instructions
L1 Load Misses	Load misses from L1 cache
L1 Store Misses	Store misses from L1 cache
L2 Load Misses	Load misses from L2 cache
L2 Store Misses	Store misses from L2 cache
Single Floating Point Intensity	Ratio of single precision FP instructions to total instructions
Double Floating Point Intensity	Ratio of double precision FP instructions to total instructions
Arithmetic Intensity	Ratio of integer arithmetic instructions to total instructions
I/O Bytes Read	Bytes read from IO
I/O Bytes Written	Bytes written to IO
Extended Page Table	Extended page table size
Memory Stall	Memory Stalls

Quartz for the two scenarios described in Section II. We use the applications with data available on all three platforms—Quartz, Ruby, and Corona—for training a supervised source model.

For Case 2, we assume that there is no common training applications' data available on platforms Quartz and Corona. However, there are common applications that run on platforms Quartz and Ruby, but not on Corona. Similarly, there are common applications that run on Ruby and Corona, but not on Quartz. In this case, Section IV-E evaluates the effectiveness of the LLM in generating counter data for TESTDFFT on Corona. For tabular data generation, we leverage the DistillGPT2 [6] LLM from the publicly available Great [2] framework.

C. Evaluation Metrics and Hyperparameter Tuning

We evaluate model performance using MSE as computed in Equation 1. We then report the average MSE across 5 trials of randomly partitioned train and test splits.

$$\text{MSE}(y, \hat{y}) = \frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N} \quad (1)$$

Here, y represents the ground truth values, $\hat{y}$ denotes the predicted values from the model, and N is the number of test samples. We use the Keras library for implementing the models and conduct hyperparameter tuning with SKLearn's Grid Search CV. High MSE indicates poor accuracy. We use feature-wise min-max scaling for all experiments.

IV. Results

A. RQ1: Impact of training a source-model using a single application

This experiment aims to evaluate how well a source model built from a single application performs when it has seen no samples from the test application. Table V presents results from using TESTDFFT as the target, and each of Laghos, Kripke, and miniVite as the source application separately.

TABLE III: Machine Specifications

Metric	Quartz	Ruby	Corona
CPU Type	Intel Xeon E5-2695 v4	Intel Xeon CLX-8276L	AMD Rome
Cores/node	36	56	48
GPU Support	No	No	Yes
GPU Type	N/A	N/A	AMD MI50
GPUs/node	N/A	N/A	8

TABLE IV: ML models and their hyperparameters.

Model	Hyper-parameters	Search range
XGBoost (XGB), Bagging, Adaboost (ADB), Random Forrest (RF), Support Vector Regressor (SVR), Decision Tree Regressor (DT), Extra Trees (ER)	max features Learning Rate n_estimators max_depth loss	$[sqrt, log2]$ $[1e^{-5}, 1e^{-1}]$ [5-100] [5-15] [squared error, huber, linear, square]
Linear Regression (LR), LassoCV (Lasso), Elastic Net CV (ENet)	cv eps gcv mode	[2-10] $[1e^{-3}, 1e^{-1}]$] [auto, svd, eigen]
KNN	neighbours metric	[1-100] [euclidean, manhattan, minkowski]

TABLE V: RQ1: Average MSE when using no adaptation during test-time using TESTDFFT as the target and each of Laghos, Kripke, and miniVite as the source model.

Model	Single-Application Source Model					
	Laghos		Kripke		miniVite	
	Ruby	Corona	Ruby	Corona	Ruby	Corona
ARD	0.143	8.761	0.0125	0.072	0.0001	13.48
ADB	0.147	0.090	0.043	0.195	0.006	13.56
ET	0.134	0.044	0.005	0.160	9.14E-06	13.58
BR	0.136	0.040	0.015	0.123	1.71E-07	13.45
SVR	0.102	0.075	0.028	0.072	1.85E-06	14.0
Lasso	0.143	2.13	0.338	0.822	9.21E-05	13.48
ENet	0.143	0.485	0.325	0.773	9.21E-05	13.48
KNN	0.144	0.032	0.011	0.152	0.0003	13.44
DT	0.163	0.068	0.073	0.222	0.024	13.70
RF	0.133	0.055	0.016	0.170	0.004	13.56
XGB	0.144	0.029	0.008	0.135	1.34E-06	14.06

From Table V, we observe that on average, using Kripke's performance as the source model yields a 66% improvement over Laghos and a 98% improvement over miniVite in predicting the relative performance of TESTDFFT. This result can be explained by Figures 2a-d where Kripke and TESTDFFT show a similar nature of correlations among their respective features and target values, which is different from that of Laghos and miniVite's. From Table V, we also observe that certain ML models are better for certain application pairs. There is no one consistent winning ML model. We also observe that it is harder to transfer knowledge from Quartz to Corona. This observation can be explained by the fact that Quartz is a CPU-based machine while Corona is a GPU-based one. Hence, their architectures are significantly diverse, which makes it difficult for the CPU-based source model to get a good predictive performance out of the box without further fine-tuning.

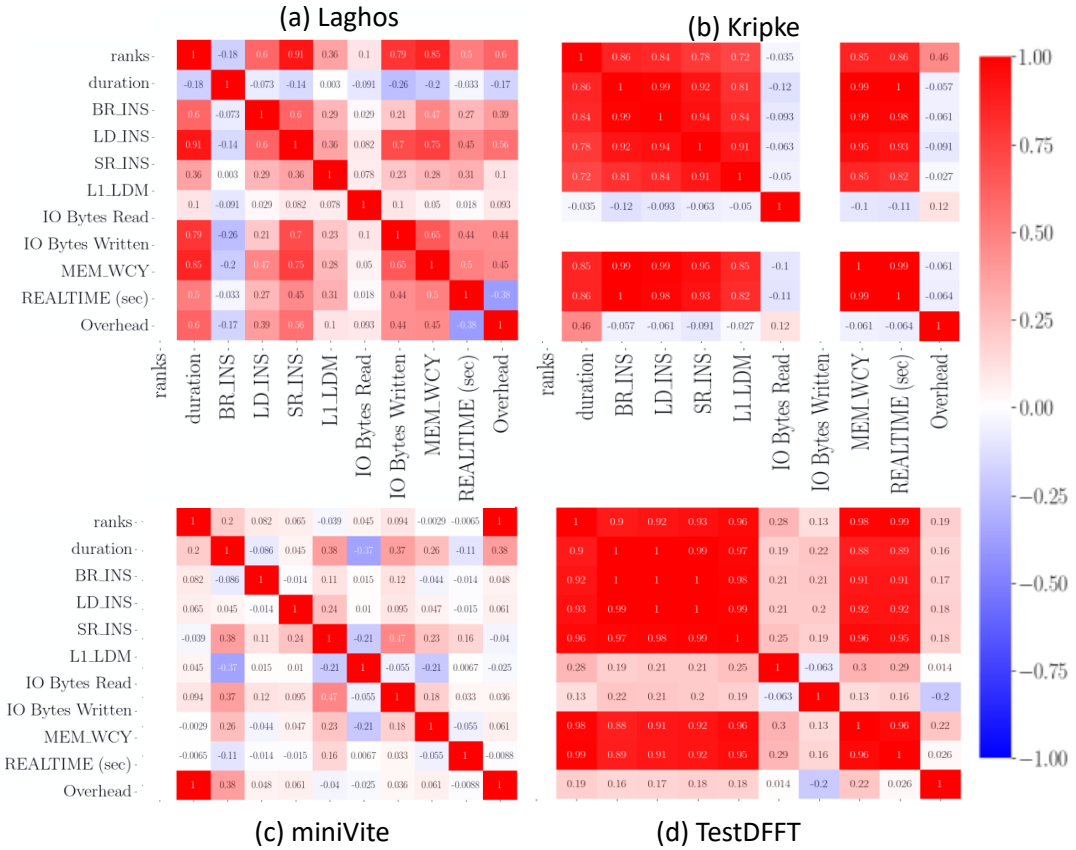


Fig. 2: Heatmap with Correlation Matrix of the hardware performance counters of (a) `Laghos`, (b) `Kripke`, (c) `miniVite`, and (d) `TESTDFFT` on `Corona`.

B. *RQ2: Impact of fine-tuning a single-source model using few-shot samples from the target application*

In this experiment, we evaluate the effectiveness of using few-shot samples to adapt single-application source models during test time. For these experiments, we use `Laghos` as the source and `TESTDFFT` as the target.

TABLE VI: RQ2: Average MSE when using a single-source model and adapting it using few-shot samples. We use `Laghos` as the source and `TESTDFFT` as the target.

Model	Applications	
	1%	
	Ruby	Corona
ARD	0.060	0.005
ADB	0.028	0.007
ET	0.024	0.005
BR	0.061	0.005
SVR	0.102	0.075
Lasso	0.031	0.006
ENet	0.032	0.006
KNN	0.021	0.006
DT	0.101	0.105
RF	0.027	0.006
XGB	0.143	0.038

From Table VI we observe that using as low as 1% samples to fine-tune the source model drastically improves its ability to generalize over zero-shot. For instance, using 1% few-shot samples can reduce error for the ET and RF models by 82% and 79%, respectively. This result supports our hypothesis that using few-shot learning to adapt the source model can improve the model's ability to generalize to new scenarios.

C. *RQ3: Impact of training a source-model using multiple applications*

In this section, we evaluate the impact of using a source model built from multiple applications' data directly without adapting it during test time. Table VII shows results from experiments using a source model built from the data of `Laghos`, `Kripke`, `miniVite`, `SW4lite`, and `AMG` and `TESTDFFT` as the target application. The results indicate that the accuracy of all models significantly improves when using a multi-source model for prediction, compared to a single-source model as shown in Table V. This improvement is observed for both `Ruby` and `Corona` across all models.

D. *RQ4: Impact of fine-tuning a multi-source model using few-shot samples from the target application*

This experiment's objective is to evaluate how well a multi-source model can generalize when adapted using a few samples during the test time. Table VIII shows the %-change in MSE when fine-tuning using 1% of samples compared to none. We use `Laghos`, `Kripke`, `miniVite`, `SW4lite`, and `AMG` to build the source and `TESTDFFT` as the target application.

Figure VIII shows that few-shot samples significantly improve prediction accuracy for some of the models compared to no adaptation, meaning the multi-source model can generalize

TABLE VII: RQ3: Impact of using multiple applications to build the source model and use it with no adaptation for TESTDFFT as the target.

Model	Platforms	
	Ruby	Corona
ARD	1.29E-05	0.01
ADB	5E-03	0.043
ET	1.39E-06	0.047
BR	4.50E-05	0.002
SVR	8.48E-06	0.032
Lasso	1.29E-05	0.000
ENet	1.29E-05	0.003
KNN	4.69E-05	0.031
DT	1E-05	0.005
RF	9E-03	0.03
XGB	5E-07	0.027

TABLE VIII: RQ4: %-Improvement in MSE of a multi-source model when fine-tuned compared to no adaptation. Only showing the subset of the ML models that shows benefit from fine-tuning a multi-source model.

Model	Applications	
	1%	
	Ruby	Corona
ARD	99.13	99.94
ADB	100.00	99.98
ET	97.17	99.87
BR	99.88	99.68
ENet	99.52	99.75

better. Specifically, by incorporating just 1% of TESTDFFT samples to fine-tune the ARD model, we observe a remarkable 99.13% improvement in accuracy for predicting relative performance on Ruby and a 99.9% improvement for Corona. However, we only observe this positive result for a subset of the ML models, not all. This result necessitates further investigation using more few-shot samples.

E. RQ5: Evaluate the predictive models under data scarcity

The objective of this experiment is to evaluate the effectiveness of using an LLM-generated performance samples for relative performance prediction. For Case 2, we assume the source platform to be Quartz, where the new application's hardware performance counter data is available. Additionally, we consider Corona as the target platform and Ruby as the intermediate platform. We consider TESTDFFT as the target; Laghos and SW4lite are run on Quartz and Ruby; Kripke and miniVite are run on Ruby and Corona.

Compared to the single-source Laghos model shown in Table V, Table IX shows that the source model built using generated data from a fine-tuned LLM can actually improve the prediction accuracy. This observation signifies that we can replace the data from Laghos using the generated data, reducing the data collection overhead.

TABLE IX: RQ5: MSE for TESTDFFT when using an LLM-generated data for building the source model.

Models	Platforms	
	Ruby	Corona
ARD	0.119	0.092
ADB	0.089	0.028
ET	0.070	0.094
BR	0.094	0.016
SVR	0.077	0.027
Ridge	0.045	0.030
Lasso	0.078	0.023
ENet	0.056	0.031
KNN	0.097	0.243
RF	0.054	0.087
XGB	0.075	0.016

F. Discussions

It is evident from Figure 2 that certain applications exhibit similar behaviors across platforms. This similarity benefits multi-source models, enabling them to perform well in zero-shot and few-shot scenarios. Single-source models, on the other hand, can suffer from data distribution shifts when applications are dissimilar. Moreover, our experiments show that fine-tuning an LLM to generate data when training samples are insufficient can be a viable solution approach with the added benefit of reduced data collection overhead.

V. Related Works

Performance prediction is a critical task in HPC. Grobelny et al. [4] propose a scalable framework for HPC performance prediction without incorporating ML methods, whereas our approach leverages ML for enhanced accuracy. Nudd et al. [11] introduce PACE, a simulation-based framework for performance and execution time prediction, but we use actual hardware performance counters for real-time predictions. Carrington et al. [3] propose a framework using signal processing techniques, while our approach employs few-shot learning and generative AI for improved accuracy. Ardalani et al. [1] use regression and ensemble models for GPU performance prediction from CPU code, focusing on GPU and CPU within the same platform, whereas we predict cross-platform relative performance. Valov et al. [15] explore transferring performance prediction models across hardware platforms but do not address data scarcity, which our approach mitigates by using few-shot samples and existing application data. Lee et al. [8] propose polynomial regression and neural networks for performance modeling, whereas we focus on relative performance prediction for new applications using few-shot learning and LLMs. Marathe et al. [9] use deep neural networks and few-shot learning for performance prediction, similar to our approach, but we further enhance the performance of transfer learning by leveraging LLMs when data is not sufficient. Kim et al. [7] and Hou et al. [5] utilize log analysis and neural networks for performance prediction. At the same time, we focus on hardware performance counter data and benchmark basic ML models, reserving advanced techniques for future work. Our method leverages few-shot samples and platform

relationships to predict relative performance effectively.

VI. Conclusions

In this work, we use a unique HPC dataset collected from various applications and platforms to evaluate the efficacy of traditional ML models for knowledge transfer. We address two scenarios: one with sufficient data and one with insufficient data. Our evaluations show that different types of ML models are well-suited for different applications and platforms. when ample labeled performance samples are available. However, when data is scarce, using few-shot learning to adapt an LLM to fill the data gap can improve the predictive models' efficacy.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-861205). This work was supported in part by LLNL LDRD projects 23-ERD-045 and 24-SI-005.

References

[1] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu. Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, page 725–737, New York, NY, USA, 2015. Association for Computing Machinery.

[2] V. Borisov, K. Sessler, T. Leemann, M. Pawelczyk, and G. Kasneci. Language models are realistic tabular data generators. In *The Eleventh International Conference on Learning Representations*, 2023.

[3] L. Carrington, A. Snavely, and N. Wolter. A performance prediction framework for scientific applications. *Future Generation Computer Systems*, 22(3):336–346, 2006.

[4] E. Grobelny, D. Bueno, I. Troxel, A. D. George, and J. S. Vetter. Fase: A framework for scalable performance prediction of hpc systems and applications. *Simulation*, 83(10):721–745, 2007.

[5] Z. Hou, S. Zhao, C. Yin, Y. Wang, J. Gu, and X. Zhou. Machine learning based performance analysis and prediction of jobs on a hpc cluster. In *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 247–252. IEEE, 2019.

[6] HuggingFace. Distilgpt2. https://huggingface.co/distilgpt2, 2019.

[7] S. Kim, A. Sim, K. Wu, S. Byna, Y. Son, and H. Eom. Towards hpc i/o performance prediction through large-scale log analysis. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 77–88, 2020.

[8] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '07, page 249–258, New York, NY, USA, 2007. Association for Computing Machinery.

[9] A. Marathe, R. Anirudh, N. Jain, A. Bhatele, J. Thiagarajan, B. Kailkhura, J.-S. Yeom, B. Rountree, and T. Gamblin. Performance modeling under resource constraints using deep transfer learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[10] D. Nichols, A. Movsesyan, J.-S. Yeom, A. Sarkar, D. Milroy, T. Patki, and A. Bhatele. Predicting cross-architecture performance of parallel programs. In *37th IEEE International Parallel Distributed Processing Symposium*, 2024.

[11] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. Pace—a toolset for the performance prediction of parallel and distributed systems. *The International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[12] G. Ozer, S. Garg, N. Davoudi, G. Poerwawinata, M. Maiterth, A. Netti, and D. Tafani. Towards a predictive energy model for hpc runtime systems using supervised learning. In U. Schwardmann, C. Boehme, D. B. Heras, V. Cardellini, E. Jeannot, A. Salis, C. Schifanella, R. R. Manumachu, D. Schwamborn, L. Ricci, O. Sangyoon, T. Gruber, L. Antonelli, and S. L. Scott, editors, *Euro-Par 2019: Parallel Processing Workshops*, pages 626–638, Cham, 2020. Springer International Publishing.

[13] L. Shao, F. Zhu, and X. Li. Transfer learning for visual categorization: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1019–1034, 2015.

[14] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, and A. Okanlawon. Improving hpc system performance by predicting job resources via supervised machine learning. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, PEARC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[15] P. Valov, J.-C. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki. Transferring performance prediction models across different hardware platforms. ICPE '17, page 39–50, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Comput. Surv.*, 53(3), june 2020.