Exploring the Utility of Graph Methods in HPC Thermal Modeling

Bruno Guindani
Department of Electronics, Information and Bioengineering
Politecnico di Milano
Milano, Italy
bruno.guindani@polimi.it

Martin Molan
Department of Electrical, Electronic and Information Engineering
Università degli Studi di Bologna
Bologna, Italy
martin.molan2@unibo.it

Andrea Bartolini
Department of Electrical, Electronic and Information Engineering
Università degli Studi di Bologna
Bologna, Italy
a.bartolini@unibo.it

Luca Benini
Department of Information Technology and Electrical Engineering
ETH Zurich
Zurich, Switzerland
lbenini@iis.ee.ethz.ch

ABSTRACT

This work critically examines several approaches to temperature prediction for High-Performance Computing (HPC) systems, focusing on component-level and holistic models. In particular, we use publicly available data from the Tier-0 Marconi100 supercomputer and propose models ranging from a room-level Graph Neural Network (GNN) spatial model to node-level models. Our results highlight the importance of correct graph structures and suggest that while graph-based models can enhance predictions in certain scenarios, node-level models remain optimal when data is abundant. These findings contribute to understanding the effectiveness of different modeling approaches in HPC thermal prediction tasks, enabling proactive management of the modeled system.

CCS CONCEPTS

• **Hardware → Temperature simulation and estimation**.

KEYWORDS

High-Performance Computing, Graph Neural Network, Thermal Modeling

ACM Reference Format:
Bruno Guindani, Martin Molan, Andrea Bartolini, and Luca Benini. 2024. Exploring the Utility of Graph Methods in HPC Thermal Modeling. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3629527.3652895

1 INTRODUCTION

High-Performance Computing (HPC) represents a pinnacle of computational capability, harnessing the power of advanced hardware and software technologies to solve complex problems with unmatched speed and precision. The price for this unparalleled performance is the high hardware and operational costs, which become even more critical with the transition to exascale. The most crucial variable cost is the energy consumption of facility infrastructure, which is not directly linked to processing [9]. The cooling of the processing elements is the main contributor to this consumption along with its associated costs [3].

Intelligent thermal monitoring and prediction systems are being introduced to minimize the cooling expense and consequently reduce the variable cost of the HPC operation. These systems vary from macro-scale, predicting the power-usage efficiency of the entire data center in connection to weather conditions, to micro-scale, modeling the thermal dynamics of the processor of the single compute node. Accurate and reliable thermal prediction models would enable more efficient utilization of HPC computing systems, such as direct integration with the scheduler [3]. Energy-aware scheduling, such as the one proposed in [3], is already a well-explored concept in the HPC domain. During periods of high cooling costs, the scheduler aims to schedule fewer and fewer intensive compute jobs on the system.

Associated with the rise in performance, HPC systems have also exploded in complexity. Exascale and pre-exascale systems have up to tens of thousands of compute nodes, each consisting of CPUs and dedicated accelerators [9]. This explosion in complexity necessitates the transition from manual analysis and domain-driven models to the introduction of machine learning models. The most recent trend in machine learning-powered predictive models for HPC is the use of graph representation and Graph Neural Networks (GNNs) [6]. HPC systems are an ideal target for GNNs as there are multiple layers of connection between logical units (compute nodes), such as physical layout or job proximity.

In this work, we critically examine the utility of the graph processing approaches for the thermal prediction use case and compare it against the domain-driven per-node model. Based on this validation, we find the best node-level thermal prediction model that can be scaled to current and future pre- and exascale HPC systems. We perform the experimental evaluation on a publicly available dataset curated by the University of Bologna [4]. All proposed models are

ICPE '24 Companion, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0445-1/24/05.
https://doi.org/10.1145/3629527.3652895

also publicly available and form the basis for future exploration of graph- and non-graph-based predictive models for HPC systems. Our code is available at https://gitlab.com/ecs-lab/hpc-thermal.

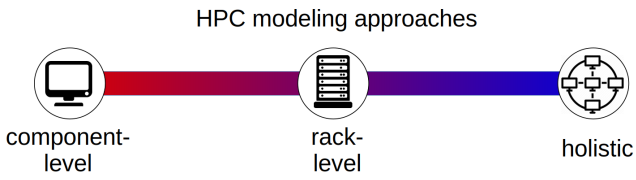


Figure 1: Taxonomy of HPC modeling approaches.

2 RELATED WORK

2.1 Machine learning approaches in HPC

There are two main approaches to building machine learning methodologies to support HPC systems: holistic and component-level models [8]. Each of them leverages the specific characteristics of HPC systems. Component-level approaches aim to build many different machine learning models such as neural networks, each tailored to a specific component or subsystem of the HPC system [7]. On the other hand, the holistic approaches aim to leverage large amounts of data produced by the HPC system to build a single, large model encompassing the whole HPC system with all its components and subsystems [1]. The midway point between room-level (holistic) and component-level models are models designed for a group of components, e.g, a set of nodes within the same compute rack [6]. We illustrate this taxonomy in Figure 1.

Component-level approaches are well-explored in the area of HPC modeling. For anomaly detection applications, it has been well-established that the best possible results come from training a separate self-supervised anomaly detection model for each compute node [7]. The common denominator of these approaches is that while the compute nodes are similar (e.g., they have the same hardware configuration), they nonetheless require models explicitly trained for those nodes. This is because each node experiences slight variations in hardware use, application utilization, and different cooling and thermal conditions. What is common across all the component-level approaches is that they use the same neural network structure, but the model for each node is trained from scratch. Using the same model structure for each of the component-level models is possible because the compute nodes in a compute room share the same hardware characteristics [7].

On the other side of the spectrum are the holistic modeling approaches (see [1, 7]). Instead of training many models with the same structure, they attempt to create a single model that provides predictions for all the components. Because of the data's large quantity and complexity, regular tabular modeling approaches cannot be utilized. Additional information in the form of a graph structure is introduced to train such models effectively. This graph structure often takes advantage of the fact that the HPC systems are organized in compute rooms, in which nodes are arranged in rows of compute racks. This physical layout can then be a basis for the graph representation of the room-level dataset. Graph-level models have proven useful for some problems, like anomaly anticipation, where they vastly outperform the component-level models [6].

2.2 Thermal modeling in HPC

In line with the taxonomy proposed in Figure 1, several thermal modeling approaches exist for the problem of compute node-level temperature prediction. These models, such as the one proposed in [2], belong to the component-level category. While there have been attempts [2, 10], [3] at building a holistic thermal model of the entire computing room, none of these approaches attempt to generate thermal predictions at the granularity of the individual compute node. Instead, existing holistic, room-level models only predict the average temperature of the entire computing room. However, for optimizing the energy efficiency of the HPC operations, as well as for energy-aware scheduling, more granular, node-level temperature predictions are needed [3].

Recent advancements in applications of graph processing methodologies, such as [6], suggest using Graph Neural Networks (GNNs) to build a holistic predictive model with a high resolution of predictions (for each individual node).

Motivated by the need for the node-level thermal prediction model, this work critically examines the component-level (node-level in our case) and the holistic modeling approaches. Specifically, it compares the same-structure, individually-trained approach (in line with [7]) against the graph-based approach inspired by [6]. To the best of our knowledge, this is the first work in the literature with a systematic focus on HPC temperature prediction via graph models.

3 METHODOLOGY

3.1 Dataset

The data used for our analysis comes from the publicly available M100 dataset [4]. It contains several features, also referred to as metrics, collected from the Tier-0 CINECA Marconi100 supercomputer over multiple years by the ExaMon HPC monitoring framework [3]. In particular, we focus on the April 2021 – September 2022 period, in which ExaMon collected all the metrics that are relevant to this work. We use this data in samples covering contiguous 15-minute time windows, each representing a snapshot of the HPC system at a certain period in time. The snapshots include, for each compute node, aggregations of data (such as average, minimum, maximum, and standard deviation) collected during the time window. The use of aggregations is necessary to have a time-uniform dataset since ExaMon samples different metrics at different frequencies. We define sample t as the snapshot starting from timestamp t and ending at 15 minutes after t. We refer to subsequent snapshots as sample $t + 1$, $t + 2$, etc.

We exclude samples that contain a proportion of missing values (NaNs) which is larger than 1%, a threshold we fixed after a comprehensive inspection of the data. This choice allows to skip snapshots with clusters of neighboring vertices with missing values, which would be difficult to impute. On the other hand, in the remaining eligible samples, we fill in NaNs with a *neighbor average* approach. That is, for a given missing value associated with a certain vertex n, metric m, and timestamp t, we collect the (non-missing) values of the m metric of the neighbors of n at time t, perform a weighted

average, and impute such average to the missing value. The weights used are the same as the corresponding edge weights in the graph model (see Section 3.3).

3.2 Prediction target

Given a snapshot associated with time t and a compute node, our prediction target variable is the *future temperature increase* measured at the node outlet, i.e., the difference between the outlet temperature measured at time at time $t + 1$ and the one measured at time t. This corresponds to a 15-minute prediction horizon, as mentioned earlier. This time horizon enables almost real-time monitoring of the HPC system, allowing timely interventions by system administrators or automatic adjustments to the cooling system if using adaptive control strategies.

We choose to predict a temperature increase rather than the raw temperature data because it contains the truly relevant information that system administrators ought to know. A large raw temperature value does not necessarily suggest a hazardous state. External factors such as weather and seasonal conditions heavily influence such value, to the point where it is not uncommon to have an average 10°C difference between both winter and summer [2]. A sudden spike in temperature instead signals a potentially hazardous state of the system, especially using a short prediction time horizon.

3.3 Models

For our temperature prediction task, we propose three models, which represent three hypotheses having increasing complexity: 1) a Ridge linear regression model for each node; 2) a Dense Neural Network (DNN) model for each node; and 3) one Graph Neural Network (GNN) model for the entire Marconi100 room.

In the graph model, we represent the Marconi100 room with an undirected weighted graph whose structure is based on the physical layout of the room. We display this layout in Figure 2. Each dot

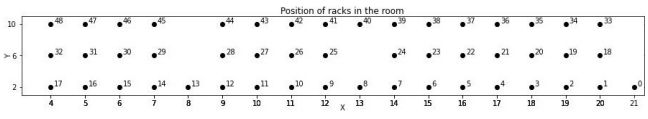


Figure 2: Spatial coordinates (in meters) of Marconi100 racks.

in the figure represents one of the 49 racks (IBM 7965-S42) in the room, each of which holds 20 compute nodes and is about 2 meters tall. The room therefore hosts a total of 980 compute nodes.

In our graph model, each vertex represents a compute node, with edges connecting it to its closest neighbors in all three spatial directions. Therefore, each vertex has at least 2 neighbors (for nodes in the corners of the room) up to 6 (for any node that is not near the sides of the room). This results in a total of 4782 weighted edges. Furthermore, edge weights are inversely proportional to the physical distance between nodes in the room. We refer to this representation as the *spatial graph* model.

We use this graph structure to perform regression using the Graph Convolutional Network (GCNConv) presented in [5]. We chose this network as it yields the best results in terms of prediction accuracy, according to preliminary experiments.

4 EXPERIMENTAL RESULTS

4.1 Experimental setting

The goal of our experiments is to assess the validity of the three models presented earlier. Specifically, for graph models, we seek to validate or reject the following two research questions:

- Does the data present a graph structure?
- Does a graph structure bring benefit to the prediction task compared to a per-node analysis?

Therefore, besides the comparison between per-node and graph models, we also conduct experiments with other graph mapping structures. Specifically, we also test a random connectivity matrix and a null connectivity matrix. In the first case, we sample random pairs of vertices and connect them through edges with randomly sampled weights. In the second case, we set all edge weights to zero, preventing the model from exchanging information between neighboring vertices.

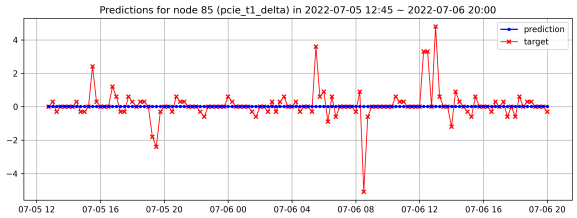
We split the collection of samples into a training set and a test set, composed of 80% and 20% of the total snapshots, respectively. We preserve the chronological ordering of snapshots during the split. This way, we simulate a real-world scenario in which we use past experience to predict the future behavior of the system. We use this split to train all models described in this work. We use the Mean Squared Error (MSE) as the loss function to minimize during training and as the validation metric for the evaluation of the test set. Relative metrics like the Mean Absolute Percentage Error (MAPE) are not suited for this prediction task, since the target variable is oftentimes very close to zero. For the graph models, we train each GCN in batches of size 20 for 10 epochs. Further details about the training process can be found in our code repository.

The baseline against which we compare all models is the trivial Last-Value Prediction (LVP), in which the prediction for a node's temperature at time $t + 1$ is equal to the temperature at time t. We use the temperature difference as the regression target, therefore the LVP is identically zero for all t. The LVP is often used as a reference benchmark for temperature in HPC settings since temperature is a slow-changing metric. In particular, the LVP represents a steady-state system approximation for the HPC system. Focusing on a performance comparison with the LVP means focusing on the meaningful temperature changes in the system.

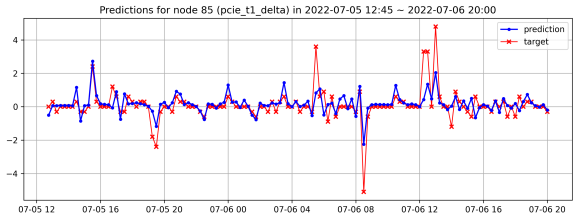
The random connectivity matrix described earlier is created by first fixing the same number of edges as in the spatial mapping (4782), then randomly sampling that many pairs of vertices. Each pair corresponds to a graph edge in the random mapping. Then, the weights for these edges are i.i.d. sampled from a uniform distribution on $(0, 10)$, which is a similar range to the original spatial mapping.

4.2 Empirical results

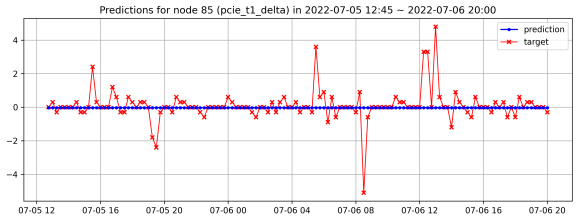
We will describe our process of improving our prediction models to improve their accuracy. The steps of this incremental process each refer to one part of Figure 3, in which we plot the true target values (in red) against the model predictions (in green) in the first portion of the test set, on a randomly chosen node. We will show that these steps lead to an overarching conclusion: *Simplicity is best.*



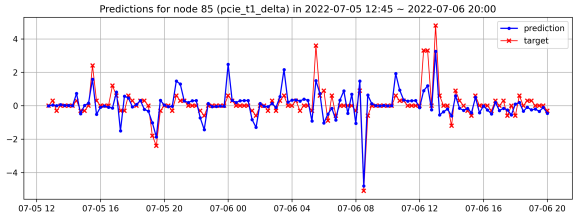
(a) Spatial graph mapping: initial high-complexity model.



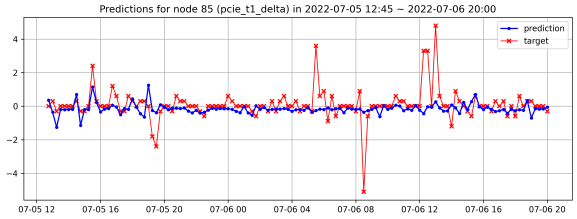
(b) Spatial graph mapping: reduced model.



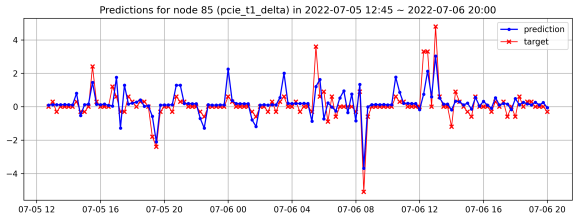
(c) Node DNN model.



(d) Ridge linear model.



(e) Random graph mapping.



(f) No graph mapping.

Figure 3: Predictions for several models for node 85.

4.2.1 Model complexity. We build our first GNN model as described in the earlier sections. In particular, we use as input all 416 aggregated node features available in the original dataset and use 10 hidden layers with regularly decreasing sizes. As shown in Figure 3a, such a model collapses to the trivial prediction of zero for all timestamps. This is also the average of all per-node temperature differences in the entire dataset, as well as the null constant value of the LVP baseline.

Since this model is unable to capture the trend of the target variable, we tried increasing the number of layers in the neural network. Still, the added complexity did not translate to an improvement in the model's predicting power. The result is nearly identical to the original model shown in Figure 3a. We then tried to head in the opposite direction by *removing* complexity from the model, in two different ways: by decreasing the number of hidden layers and by considering a smaller subset of all available features. In particular, out of the original 416 aggregated node features, we only choose to keep three that we deem the most relevant and essential. Two of these features are used twice, once representing information coming from snapshot $t-1$, and once from snapshot t. This results in five features being used in the GCN models. We list such features in Table 1. The underlying physical processes of heating suggest

name	description
`pcie_avg_t-1`	average outlet temperature in snapshot $t-1$
`pcie_t0`	outlet temperature at time instant t
`total_power_avg_t-1`	average node power consumption in snapshot $t-1$
`total_power_avg_t0`	average node power consumption in snapshot t
`ambient_avg_t-1`	average inlet temperature in snapshot $t-1$

Table 1: Reduced subset of node features.

that given temperature information from time $t-1$ to t, as well as the total power consumption from time $t-1$ to $t+1$, we should be able to infer the temperature at time $t+1$. In Figure 3b, we can see that the changes bring positive benefits to the model, which is now capable of capturing the fluctuations of the next temperature difference. We may attribute the worse performance when using all available features to the fact that most of the removed ones are likely irrelevant when inferring temperature, therefore representing a source of noise.

Simplicity also has a positive impact concerning the GNN model complexity. Specifically, the best-performing model only has one hidden GCN layer (plus one input and one output layer). As mentioned, increasing the number of hidden layers does not improve the model performance, and can even have a negative impact when adding too many. We can explain this phenomenon by the determinism of the underlying physical process.

Simplicity also translates to a lack of complexity in the chosen GNN model. As previously mentioned, we tested several network models, and GCN [5] stood out as yielding the best prediction performance. This is one of the simplest models available in the literature (see, e.g., [11]).

4.2.2 Per-node models. We now build individual Dense Neural Network (DNN) models for each individual compute node, by using the same layer structure as the GNN, except with GCN layers swapped out for traditional fully connected layers. We also use as

model	subplot	MSE
LVP	–	2.260
spatial graph (initial)	(a)	2.261
spatial graph (reduced)	(b)	1.667
node DNN	(c)	2.261
Ridge	(d)	1.103
random graph	(e)	2.436
no graph	(f)	1.453

Table 2: Average test-set Mean Squared Errors for several models.

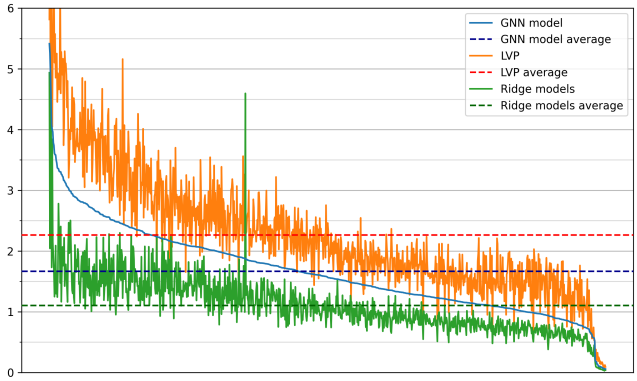
input the same reduced subset of the five features described earlier. However, as depicted in Figure 3c, this model also degenerates to a constant prediction. This result is consistent with our observation about complexity. Indeed, an entire neural network for an individual compute node is arguably more complex than a portion of a single graph network modeling all nodes at once – in other words, 980 independent DNNs bring more complexity than one GNN.

Finally, we attempt to simplify our assumptions even further by training one Ridge linear regression model for each compute node. Similarly to the DNN per-node model, we use the reduced subset of features. As exemplified in Figure 3d, not only does this model not collapse to a constant, but it also outperforms the best GNN model trained so far, i.e., the reduced one (Figure 3b). This is another piece of evidence suggesting that less complex models are best suited to this prediction task.

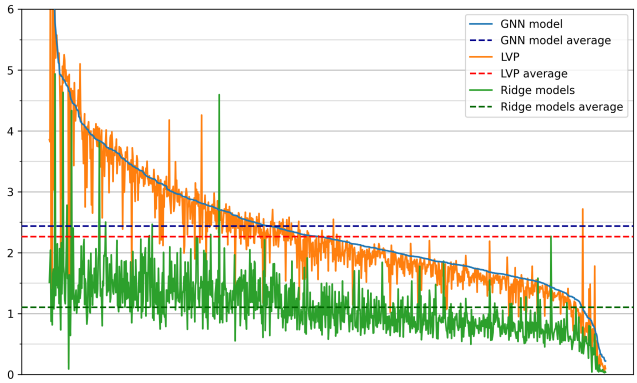
4.2.3 Model comparison. We show the full extent of the comparison between the GNN model and the linear model in Figure 4a. Specifically, each point on the blue line represents the average test-set MSE of the GNN predictions on an individual node. These points are sorted in decreasing order. The figure also shows the corresponding MSE for both the LVP baseline (in orange) and the Ridge linear model (in green), by keeping the same order of nodes used for the GNN errors. In other words, vertically aligned points refer to the same compute node. Finally, the horizontal dashed lines represent the average MSEs of the three models across all compute nodes. Figure 4a shows that Ridge models outperform the GNN model in nearly all individual nodes. Nonetheless, both models score lower errors than the LVP baseline.

4.2.4 Graph structure validation. We now report results for both the random and the zero-weight graph mapping. Their predictions on the chosen test-set window are displayed in Figures 3e and 3f, while the errors for all compute nodes are in Figures 4b and 4c, respectively. The random graph model does not degenerate to a constant either, but its predictions are less precise than the reduced spatial model – in fact, they turn out to be worse than even the trivial LVP baseline. This is most evident from Figure 4b: the curve of the GNN model error (in blue) lies above the LVP errors (in orange) for the majority of the nodes. On the contrary, the zero-weight graph model shows an improvement compared to the spatial model but still falls short of the linear regression model (as is clear from Figure 4c).

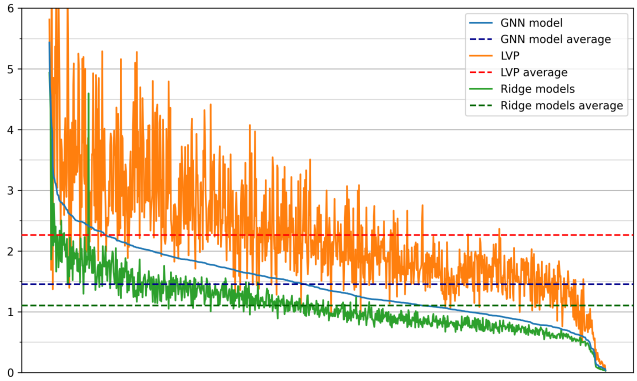
We show the average test-set MSEs of all models in Table 2. These figures are consistent with the rest of our analysis. The initial



(a) Spatial graph mapping: reduced model.



(b) Random graph mapping.



(c) No graph mapping.

Figure 4: Sorted Mean Squared Errors for several models.

spatial graph model (a) and the node DNN model (c) are very close to being constant models, thus they have the same error as the LVP baseline. The random graph model (e) performs worse than the baseline and the reduced spatial graph model (b) achieves worse performance than the zero-weight graph model (f), which in turn is worse than the per-node linear model (d).

4.2.5 Graph vs per-node models. These results lead us to the following considerations. The fact that the best models (in terms of

MSE) do not use graph information, shows that a graph structure is not needed to perform prediction on the M100 data, nor does a graph structure necessarily improve results in a regression setting. However, an inappropriate graph structure (such as the random one in (e)) can still influence the model's predictive capability in a negative way. This indicates that the data may still present graph-like patterns, which do not emerge when using a random graph structure.

5 DISCUSSION

This work critically examines the node-level and holistic modeling approaches for thermal prediction in HPC systems. Based on the experimental evaluation, the best-performing holistic approach is the spatial GNN which is trained to predict temperature changes in connection to the current temperature. It beats the baseline and achieves good prediction results. We also observe the importance of a correct graph structure: if the graph structure is shuffled, the prediction performance drops significantly, becoming even worse than the trivial baseline.

However, the holistic approach performs sub-optimally compared to the node-level approach. A simple node-level model based on domain-based feature engineering outperforms the graph model. This contrasts with the recent result in [6] on the same dataset, where graph-based models severely outperformed the node-level models on anomaly prediction tasks.

The difference between the anomaly prediction and the temperature prediction cases is the availability of labels. Anomaly prediction is an unbalanced classification task; per-node models do not have sufficient data to learn non-trivial predictions. In the temperature prediction case, however, the future temperature information is equally abundant in both the graph and node-level cases. The comprehensive study conducted in this paper thus suggests that when sufficient data is available, simple node-based models outperform more complex graph models.

Surprising results presented in this study nicely complement the current preliminary results on GNN applications in the HPC domain. While graphs are a powerful computational tool that unlocks the possibilities of fulfilling novel predictive tasks, their utility lies mainly in augmenting other models' poor or missing data, such as anomaly prediction cases. However, when data is abundant, the classic per-node and per-component modeling approaches still give the optimal results.

ACKNOWLEDGMENTS

This research was partly supported by the HE EU Graph-Massivizer project (g.a. 101093202). This work was also supported by the Italian Ministry of University and Research (MUR) under the National Recovery and Resilience Plan (PNRR) and by the European Union (EU) under the NextGenerationEU project. Finally, we thank CINECA for their collaboration and access to their machines.

REFERENCES

[1] Aksar, B., Zhang, Y., and Ates, E. e. a. Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems. In *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings 36* (2021), Springer, pp. 195–214.

[2] Ardebili, M. S., Bartolini, A., Acquaviva, A., and Benini, L. Rule-based thermal anomaly detection for tier-0 hpc systems. In *International Conference on High Performance Computing* (2022), Springer, pp. 262–276.

[3] Borghesi, A., Conficoni, C., Lombardi, M., and Bartolini, A. Ms3: A mediterranean-stile job scheduler for supercomputers-do less when it's too hot! In *2015 International Conference on High Performance Computing & Simulation (HPCS)* (2015), IEEE, pp. 88–95.

[4] Borghesi, A., Di Santi, C., Molan, M., Ardebili, M. S., Mauri, A., Guarrasi, M., Galetti, D., Cestari, M., Barchi, F., Benini, L., et al. M100 exadata: a data collection campaign on the cineca's marconi100 tier-0 supercomputer. *Scientific Data 10*, 1 (2023), 288.

[5] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations* (2017).

[6] Molan, M., Ahmed Khan, J., Borghesi, A., and Bartolini, A. Graph neural networks for anomaly anticipation in hpc systems. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering* (2023), pp. 239–244.

[7] Molan, M., Borghesi, A., Cesarini, D., Benini, L., and Bartolini, A. Ruad: Unsupervised anomaly detection in hpc systems. *Future Generation Computer Systems 141* (2023), 542–554.

[8] Ott, M., Shin, W., Bourassa, N., Wilde, T., Ceballos, S., Romanus, M., and Bates, N. Global experiences with hpc operational data measurement, collection and analysis. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)* (2020), IEEE, pp. 499–508.

[9] Shin, W., Oles, V., Karimi, A. M., Ellis, J. A., and Wang, F. Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2021), pp. 1–14.

[10] Taneja, S., Zhou, Y., and Qin, X. Thermal benchmarking and modeling for hpc using big data applications. *Future Generation Computer Systems 87* (2018), 372–381.

[11] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI open 1* (2020), 57–81.