I/O-ETEM: An I/O-Aware Approach for Estimating Execution Time of Machine Learning Workloads

Elham Adibi, Mohammadamin Ajdari, Pouria Arefijamal, Amirsaeed Ahmadi-Tonekaboni, and Hossein Asadi

***Abstract*—Training *Machine Learning* (ML) models commonly rely on *High-Performance Computing* (HPC) centers or cloud servers that accommodate compute nodes with powerful resources. Users tend to enhance the accuracy of ML applications by continuous training, after model refinements, and dataset size increase. With such application changes, and heterogeneity of HPC nodes, knowing each job execution time in advance (i.e., *prediction*) is necessary for efficient job scheduling. We observe that I/O accesses highly influence the executing time of modern ML applications. Unfortunately, existing studies on estimating job execution time either (a) rely on overestimated user declared time, or (b) predict execution time mainly based on compute resources (ignoring I/O or storage effects), and use complex deep learning models for this purpose. In this paper, we propose a simple, yet effective method for predicting the execution time of ML training. Our approach explicitly accounts for I/O accesses as a critical factor. Our method combines (a) partial application execution & monitoring, (b) analytical modeling leveraging ML application characteristics, (c) dynamic re-estimation, and (d) simplified history-based analysis. Our evaluation on a number of *Convolutional Neural Networks* (CNNs) and Transformer models show that our proposed method predicts the execution time accurately (i.e., with error less than 8% for most cases) compared to actual execution.**

***Index Terms*—Machine Learning, Model Training, Execution Time Prediction, High-Performance Computing, Job Scheduling.**

I. Introduction

Training *Machine Learning* (ML) models are known for being resource-hungry and time-consuming. With their growing popularity, such workloads are becoming widely deployed on *High Performance Computing* (HPC) centers or large cloud providers due to availability of fast compute- and storage- resources at decent costs.

Accurate *prediction of a job execution time*, prior to complete execution, is an important part of efficient scheduling on HPC nodes. A recent study shows that 85% of execution time in modern ML workloads are consumed by I/O accesses [1]. However, *the execution time estimation logic in existing HPC schedulers do not treat I/O as a first-grade component; thus are inherently inaccurate.*

We observe (a) that ML applications are evolving to become more accurate through gradually (a) training with other, possibly larger datasets, or (b) through adding additional layers or model refinements (e.g., converting ResNet-18 to ResNet-152 by roughly adding 134 more layers). *We reveal that changing the datasets in the past had little impact on training time due to the compact dataset sizes. In modern applications, however, a sample 3× larger datasets may lead to linearly 3× larger training time.*

We also observe (b) that *heterogeneity in storage resources of HPC nodes, which is required for execution of different user applications with different budgets, significantly change the training time for modern ML applications on a specific HPC node compared to another node. However, such heterogeneity had little effect in the past with smaller datasets*. For example, our sample experiment shows that training BEity transformer on Imagenet dataset on a node with available local-DRAM cache provides 4.8× shorter training time compared to typical compute nodes with no local cache accessing data on network-attached *Hard Disk Drives* (HDDs).

On the contrary to our observations, traditional schedulers rely on user-declared execution time, which are commonly overestimated. Some researchers have introduced history-based predictors (relying on neural networks), which completely ignore I/O-related information [2], or predict only I/O bursts (e.g., as in [3]) not the application execution time. Some studies capture I/O hints from user script metadata (as in Prionn [4]), and build a complex ML model to predict application execution time. Such approaches are application-agnostic (*no special consideration* for ML training applications), and do not deal with I/O or storage infrastructure explicitly. Other studies perform short runs to infer full execution time of applications. Such approaches focus on CPU-only operations [5], or GPU compute operations [6], [7], ignoring I/O overheads. In summary, existing studies (a) mostly ignore I/O, (b) are typically complex without providing any simple or quick relations of application execution time with its dataset, (c) do *not* focus on important domain of improving ML applications (e.g., model refinement and dataset changes) and also do *not* consider storage heterogeneity in HPC nodes, which are the goals of this paper.

In this paper, we propose a simple, yet effective **I/O**-aware **E**xecution **T**ime **E**stimator for **M**L jobs (*I/O-ETEM*) with four key ideas. *First*, for the changes in ML application dataset size, I/O-ETEM runs the application (or a representative part) only once to build a reference I/O model. Then, by using our intuitive scaling formula, I/O-ETEM estimates the training time for any new datasets (of similar type). *Second*, for ML model refinements or storage infrastructure changes, we propose *Partial Run & I/O Monitor*, which builds a reference I/O model based on a partial execution and predicts total execution time by our proposed scaling formulas. *Third*, we adopt an I/O contention modeling and dynamic phase change detection for accurate predictions at any moment with multi-tenant environments. *Fourth*, we use a database-matching (a simplistic history-based analysis) to detect previously run ML applications on exactly same environments and provide immediate predictions without any partial runs.

We implemented and evaluated I/O-ETEM on a real system. Our evaluation on a number of *Convolutional Neural Networks* (CNNs) and Transformer applications shows that I/O-ETEM accurately predicts the execution time under the following scenarios: a) less than 6.2% error for model refinements from `ResNet-18` to `ResNet-152`, b) an average 14% error for dataset size change, c) 4-8% error for storage infrastructure changes, and d) less than 2% error for I/O contention modeling. I/O-ETEM is accurate and its simplicity *eases the deployability*, and also enables architects to *quickly understand the behavior of an ML training*.

II. Motivation on Execution Time Prediction

***We observe** that heterogeneity of storage infrastructure in HPC compute nodes and repeated execution of ML training with larger datasets, changing application parameters (or ML model refinements) **exhibit up to an order of magnitude different training time** compared to previous run on another node, or with another dataset size; thus, a necessity to predict its ML training time well in-advance to schedule*

Hossein Asadi is the corresponding author. All authors are affiliated with Sharif University of Technology, Tehran, Iran. Mohammadamin Ajdari is additionally affiliated with HPDS Research, Tehran, Iran. (e-mail: e.adibi, m.ajdari, pouria.arefijamal, amirsaeed.ahmadi99, asadi@sharif.edu). This work was partially funded in part by the *Iran National Science Foundation* (INSF) under Grant 4025937.

on a proper node. **First**, different storage infrastructure for one HPC compute node compared to the other node means different speed of accessing the training dataset per node. Our sample experiment shows that training BEity transformer on `ImageNet` dataset on a node with available local-DRAM cache provides 4.8× shorter training time compared to typical nodes with no local cache (accessing data on network-attached HDDs). **Second**, changing the training dataset in the past had *no* major effect on training time. However, with emerging large datasets and significant dataset loading overheads compared to computation overheads, switching from `ImageNet` to a 3× larger datasets almost triples the training time, making the dataset size an important factor in job time prediction. **Third**, in a well-utilized HPC center, most nodes are running some jobs while new jobs may arrive. In this case, *predicting when each currently running job finishes* is necessary for efficient node allocation for incoming jobs.

A. Limitation of Existing Work

Existing studies can be classified into two categories: (a) **history** analysis and (b) **runtime** analysis. The former considers the jobs of the past and typically applies a complex ML model to find similar patterns. Some of these predictors rely on manually extracted features from job attributes to predict execution time [8], or to predict I/O bursts [3]. A few recent approaches try to identify complex patterns directly from the raw data of user scripts by training a neural network model, but these predictors either ignore I/O-related information [2] or may have implicitly included minimal I/O information available in HPC user submission scripts, ignoring storage infrastructure effects (e.g., as in [4]). Furthermore, they have high complexity and require additional resources for building and using the prediction model. Note that some studies build data-hungry statistical models to predict storage access performance [9], but are both complex and also focus on filesystem benchmarks (not ML applications).

Some studies have proposed **runtime** analysis for predicting execution time based on performance models focused on compute resources (no I/O consideration) [6]. As an example, Habitat [7] runs the application on a specific GPU model, then predicts its execution time on a different GPU model. Some execute a portion of an application, compare its performance model to previously run benchmarks, then estimate its full execution time [5]. These methods work well for applications similar to specific benchmarks (primarily CPU-based), and partly rely on static code analysis. As such, they may not be suitable for modern GPU-centric or rapidly evolving ML applications. *All these limitations demand designing a new approach to be **accurate and simple** in predicting execution of ML applications in common HPC use cases: (1) dataset size change, (2) storage infrastructure change, (3) ML model refinements, and (4) executing same ML application again.*

III. Proposed Method

We propose an **I/O**-aware **E**xecution **T**ime **E**stimator for **ML** jobs (*I/O-ETEM*) with *four key ideas*: **(a)** an *intuitive, linear scaling formula* that enables immediate prediction of an ML execution time by applying simple arithmetic (for cases of changing the training dataset size), **(b)** *Partial Run & Estimate*, which requires running a small portion of ML application to predict its whole execution (for ML model refinements, or storage infrastructure changes), **(c)** simple but effective *I/O contention modeling* in HPC clusters and dynamic phase change detection (for accurate predictions at any moment), **(d)** adopting a much simpler history-tracking idea compared to state-of-the-art by using a *database-matching* approach with (ML application, inputs, infrastructure and conditions) as a key and *(execution time)* as value, to immediately predict future execution time of any previously run ML applications under same conditions.

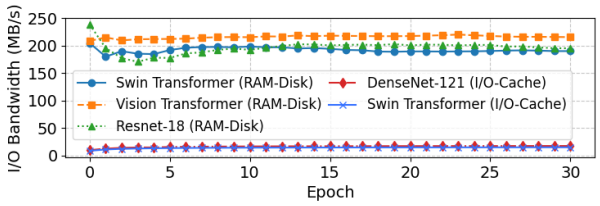

Fig. 1. I/O bandwidth stabilizes after 3 to 5 epochs of training

A. I/O-ETEM Basic: Dataset-Aware Execution Modeling

The first important component that we include in I/O-ETEM, which is usually absent in previous works is *Dataset Size* of an ML application. I/O-ETEM introduces a dataset-aware execution model that estimates ML training time based on two assumptions: (1) I/O-time constituting most of the execution time (e.g., 85% as stated in [1]) or all of the execution time (in pipelined, balanced compute- and storage-resources), and (2) a known, stable storage bandwidth. Under these conditions, execution time can be estimated as:

$$\text{Execution Time} = \frac{N \times \text{Dataset Size}}{\sum_{i=1}^{K} \text{Storage Bandwidth}_i} \quad (1)$$

Where N is the number of training epochs, and K is the number of nodes involved in the training of a desired ML application. Our intuitive model enables immediate estimation of job execution time when only the dataset size changes, and it implicitly considers all system-level effects such as file systems, CPU/GPU model, ML model, and storage infrastructure as long as they do not change. Our analysis of GPU clusters of a major HPC center (ThetaHPC [10]) reveals that 92.9% of jobs executed in 2023 used up to three nodes (and single-node runs as most prevalent). Thus, our goal in this paper is these common small-scale training jobs, not heavily distributed training (with significantly different behaviors).

Limitation. If major system components change (e.g., storage infrastructure due to HPC node migration or ML model parameters for tuning), I/O bandwidth usage may change (*invalidating assumption 2*). This demands additional key ideas or optimizations for I/O-ETEM.

B. Optimization 1: Adapting to Model and Infrastructure Changes

To handle model refinements (parameter tuning or ML layer changing), change in storage infrastructure, or significant change in dataset type (not size), we propose *Partial Run and Estimate*. This key idea is based on two insights: (a) ML training has a repetitive nature, for example, reading the training datasets multiple times (i.e., multi-epoch training) to increase the model accuracy, (b) after a small number of epochs (e.g., *three* epochs), the average resource utilization (especially, storage bandwidth usage) stabilizes and becomes almost constant (Fig. 1). In this regard, we propose running the workload for predefined small threshold, till the storage bandwidth usage has minimal variations. Then using Formula 1, and the amount of remaining data for training (i.e., datasets), and average storage bandwidth, the total execution time can be accurately estimated. Algorithm 1 shows the pseudo-code for this operation. Note that the simplicity, and intuitiveness of Formula 1, and Optimization 1 enables *system architects to understand an ML application behavior quickly* for applying some manual system tuning. This is a bonus benefit, in addition to the main benefit of our approach, which is to enable efficient HPC scheduling.

C. Optimization 2: Handling System and Workload Variability

We address dynamic changes in system or infrastructure behavior (e.g., changes in available storage bandwidth over time), in addition to application behavior changes (e.g., from data augmentation or prefetching) by proposing two mechanisms. First, we apply an **I/O**

Contention Modeling, inspired by a prior work [11] to model the possible contention in the usage of shared resources (i.e., modeling available shared storage bandwidth for a specific compute node at the time of job execution). The shared resource has a fixed maximum bandwidth capacity, named *PeakBW*. Before a job execution on a specific node starts, I/O-ETEM subtracts the bandwidth already allocated to earlier compute nodes (sharing same storage backend) from *PeakBW* value to determine the *RemainingBW* available for the current node(s). We then define an *Interference Factor (IF)* as the ratio of *RemainingBW* to *OptimalBW*. OptimalBW is the storage bandwidth observed during the application partial run measurement. If IF = 1, no contention has occurred. If IF < 1, I/O contention has occurred, and I/O-ETEM re-estimates job execution time based on the new storage bandwidth (i.e., *RemainingBW*). If *IF* > 1 (i.e., the available bandwidth is more than the application can use), I/O-ETEM uses *OptimalBW* (that application would consume) for execution time estimations and not the *RemainingBW*. Note that I/O-ETEM also has **dynamic phase change detection** in applications by monitoring bandwidth usage across epochs and triggering time re-estimation when a significant deviation (e.g., $\pm 10\%$) is detected.

Algorithm 1 I/O-ETEM Partial Run and Estimate

Require: Dataset size D, Number of epochs N, Threshold epoch T, Job J, Storage type S, Number of nodes K
Ensure: Estimated execution time for N epochs: $estimate$

$total\_bandwidth \leftarrow 0$
$io\_bandwidth \leftarrow 0$
for $epoch \leftarrow 1$ to T **do**
    Run J on K nodes with storage S for $epoch$
    Measure bw_{epoch} for each node in $epoch$
    $total\_bandwidth \leftarrow total\_bandwidth + \sum_{i=1}^{K} bw_{epoch,i}$
$io\_bandwidth \leftarrow total\_bandwidth/T$
$estimate \leftarrow (D \times N)/io\_bandwidth$
return $estimate$

D. Overall Architecture and Implementation

Fig. 2 shows the high-level workflow of I/O-ETEM. The system begins by receiving job execution information from user-provided HPC scripts. If (a) the same workload with identical conditions has been previously run, the execution time recorded in the database would be used as estimate. (b) If the workload and dataset type match, but the dataset size differs, I/O-ETEM applies scaling formulas (e.g., based on I/O bandwidth) to estimate the new execution time. (c) If the workload or hardware infrastructure is new, I/O-ETEM applies Partial Run to estimate the total execution time. After estimating the execution time, I/O-ETEM accounts for potential dynamic changes and I/O contentions in infrastructure, re-estimates the execution time if necessary, and updates the workload profile in the database.

We implemented I/O-ETEM in *330 lines of Python code*. Our implementation of I/O-ETEM clears the page cache upon each run, and starts two monitoring tools to build performance model. I/O-ETEM uses external tools of `iostat` (for disk-bandwidth usage) and `blktrace` (for total I/O read/write operations), with negligible resource usage (less than a CPU core and 1-2GB of DRAM). After the completion of a partial run of an application, I/O-ETEM records the execution time, applies the performance model, and our scaling formulas to predict the total execution time of an application.

IV. Evaluation Methodology and Experimental Results

We evaluate I/O-ETEM on a diverse set of ML workloads ranging from CNN architectures (e.g., `ResNet-18` to `ResNet-152`, and `Densenet-121`) to transformers (e.g., `EfficientFormer`, `Vit-B/32`). We use the ImageNet ILSVRC2012 dataset, which

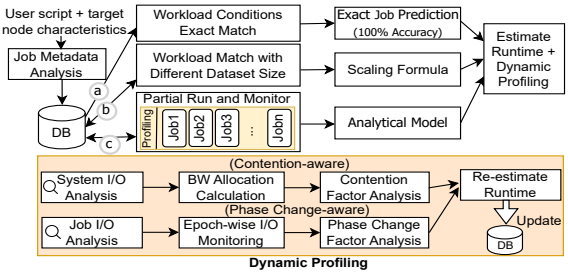

Fig. 2. Summary of our proposed method (I/O-ETEM)

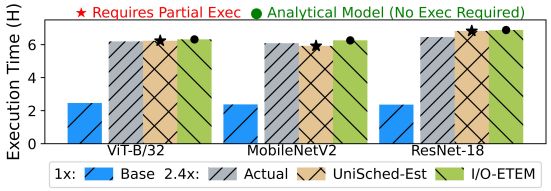

Fig. 3. Accurate prediction of ML training time by I/O-ETEM analytical model for cases with 2.4× larger training dataset size vs. *Unisched-Est* which requires partial runs every time

includes 1,000 diverse classes and is 146GB, for our experimental analysis. To explore different dataset sizes, we employ data augmentation and modulo indexing, while stochastic downsampling is applied to mitigate overfitting and maintain dataset diversity. Also, a reduced number of classes is selected for training to reduce dataset size.

We conduct experiments on a server equipped with an Intel Xeon E5-2696v4 CPU, 48GB RAM, dual GPUs (RTX 4090/2070 Super), and separate storage types including *Storage Area Network* (SAN) using HDDs and SSDs, and node-local 200GB DRAM cache setup using OpenCAS. We validate I/O-ETEM on a single node (due to popularity of jobs in major HPC centers [10]); our selected experiments on two-node setup also show similar I/O-ETEM accuracy. We compare I/O-ETEM predictions against actual training times, in addition to a recent existing work (i.e., *Unisched* [6]). *Unisched* monitors a 5-minute run for every job, counts the number of training iterations, and linearly scales to full iterations. We simulate *Unisched* Estimator (*Unisched-Est*) and consider partial runs to be as long as an epoch (i.e., 20-30 minutes in our experiments). Such choice may even increase *Unisched-Est* accuracy.

A. Predicting Effect of Training Dataset Size Change

Unisched-Est provides less than 5% error when the dataset size changes and achieves this at the cost of *partial runs of the application every time* the dataset changes. However, I/O-ETEM does not require any additional runs, and its *analytical model* provides *immediate prediction* of ML training with different dataset sizes at only 14% average error (Fig. 3). By partially running an ML application once (shown as *Base* in the figure), I/O-ETEM records I/O bandwidth and uses to calculate the execution for any future dataset size with the same type and structure (in this case, 2.4× larger dataset). Such modeling is simple and fast, and as expected, leads to suitable prediction (unlike existing methods that completely ignore dataset size or require complex model building).

B. Predicting Effect of Storage Infrastructure Change

I/O-ETEM achieves high accuracy through partial runs and I/O monitoring (first three epochs), with prediction errors of *4% for SAN HDD* and *8% for local-cache nodes*; *Unisched-Est* has higher error for SAN HDD, and *suffers from up to 73% error for local-cache nodes* (Fig. 4). Cached environments have dynamic cache hit-rate changes, and the I/O-ETEM special I/O considerations capture such behavior, but *Unisched-Est* short runs (suitable for compute-only or static environments) cannot capture such changes at all.

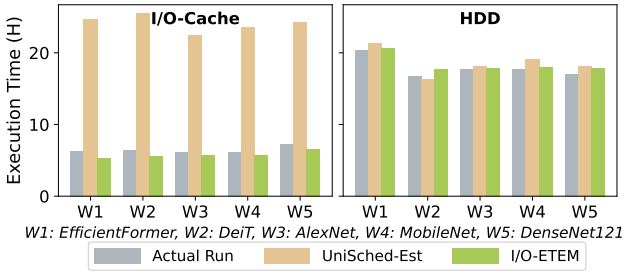

Fig. 4. *Unisched-Est* and I/O-ETEM Prediction across different storage setups

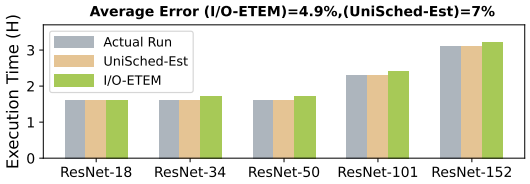

Fig. 5. Prediction accuracy of training time by *Unisched-Est* and I/O-ETEM in case of ML model refinements (case study on `ResNet`)

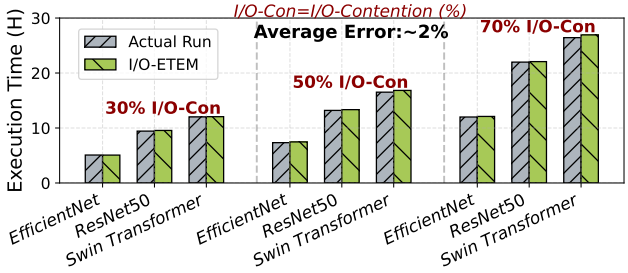

Fig. 6. I/O-ETEM re-estimates execution time under varying I/O contention

C. Predicting Effect of ML Model Refinement

I/O-ETEM's partial run & I/O monitor captures the behavior of complete modification of ML model (or refinements), thus predicting training time with little average error of 4.9% on five different ResNet architectures (Fig. 5). *Unisched-Est* has similar error of 7% if the storage environment is static. Converting `ResNet-18` to `ResNet-152` (with almost 130 more layers) significantly changes the application behavior (increasing Cuda memory and Cuda core usage from almost 25% to 95%, and reducing storage 90% to below 80%). Note that unlike I/O-ETEM, *it is almost impossible for most existing history-based predictors to estimate execution time after ML model refinement* because the model is not similar to any previous run. *Unisched-Est* may use partial runs to predict training time after model refinement, but its accuracy heavily depends on storage infrastructure.

D. Prediction Under I/O Bandwidth Contention

I/O-ETEM accurately re-estimates application execution time with negligible error when I/O contention happens (Fig. 6). We evaluated three ML models (e.g., `ResNet-50`, `Swin Transformer`, `EfficientNet`) across four scenarios: with no I/O contention, with 30%, 50%, and 70% I/O contention levels. Upon I/O contention detection, I/O-ETEM calculates and assigns proper available storage bandwidth to the application and re-scales the execution time with less than 2% error. *Unisched* does not consider I/O bandwidth, thus has no analytical re-estimations if contentions happen.

E. Sensitivity Analysis on Number of Epochs

I/O-ETEM prediction in the worst case (i.e., on local cache with fluctuating I/O bandwidth) has little error. As the number of epochs increases, such error exponentially decreases; however, Unisched-Est not only has higher error at first, increasing the number of epochs significantly increases its prediction error (Fig. 7). We run the training on `DenseNet-121` model for 5 to 30 epochs. I/O-ETEM uses up to 3 epochs (for partial execution) and has 12% prediction error for 5 epochs, because the cache behavior changes from one epoch to next one till becoming stable. On the other hand, *Unisched-Est* one-epoch partial run results in 37% error for 5 epochs. By increasing the number of epochs (common in ML training), application behavior (i.e., cache behavior) becomes more stable, leading to less than 0.2% error for 30 epochs, and expected less than 2% error for 100 epochs when using I/O-ETEM. However, *Unisched-Est* shows a reverse behavior, reaching prediction error of 45% with 100 epochs.

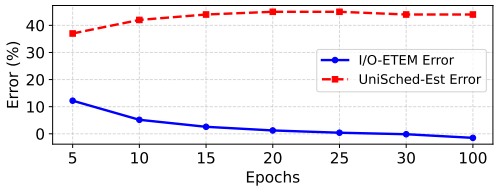

Fig. 7. I/O-ETEM enhancing prediction accuracy (lower error), and *Unisched-Est* degrading accuracy after increasing epochs (training `DenseNet121`)

V. Conclusion and Discussion

I/O-ETEM is applicable to any single- or multi-node ML training as long as storage accesses are dominant. I/O-ETEM analytical model, especially the relation of training time with dataset size and I/O-bandwidth is a new contribution over prior work. We expect its analytical model to work for both short and long training jobs, and also from small-scale to large-scale training. I/O-ETEM partial-run accurately captures ML training behavior (unlike prior work such as *Unisched* [6]), and has negligible overhead with long training jobs (e.g., less than 5% of training time), and is best with small number of nodes involved. However, for very short jobs (e.g., less than 3-epoch training), or for heavily distributed training with many nodes, its partial run overheads could be non-negligible.

References

[1] R. Böhringer, N. Dryden, T. Ben-Nun, and T. Hoefler, "Clairvoyant prefetching for distributed machine learning i/o," *Int. Conf. for High Performance Computing, Networking, Storage, & Analysis*, 2021.

[2] Y. Gao, X. Gu, H. Zhang, H. Lin, and M. Yang, "Runtime performance prediction for deep learning models with graph neural network," in *IEEE/ACM Int. Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2023, pp. 368–380.

[3] E. Saeedizade, R. Taheri, and E. Arslan, "I/o burst prediction for hpc clusters using darshan logs," in *2023 IEEE 19th International Conference on e-Science (e-Science)*. IEEE, 2023, pp. 1–10.

[4] M. R. Wyatt, S. Herbein, T. Gamblin, A. Moody, D. H. Ahn, and M. Taufer, "Prionn: Predicting runtime and io using neural networks," in *International Conference on Parallel Processing*, 2018, pp. 1–12.

[5] A. Jajoo, Y. C. Hu, X. Lin, and N. Deng, "Slearn: A case for task sampling based learning for cluster job scheduling," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 2664–2680, 2022.

[6] W. Gao, Z. Ye, P. Sun, T. Zhang, and Y. Wen, "Unisched: A unified scheduler for deep learning training jobs with different user demands," *IEEE Transactions on Computers*, vol. 73, no. 6, pp. 1500–1515, 2024.

[7] X. Y. Geoffrey, Y. Gao, P. Golikov, and G. Pekhimenko, "Habitat: A {Runtime-Based} computational performance predictor for deep neural network training," in *USENIX Annual Technical Conf. (ATC)*, 2021.

[8] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[9] J. Marquez and O. H. Mondragon, "Modelling the impact of cloud storage heterogeneity on hpc application performance," *Computation*, vol. 12, no. 7, p. 150, 2024.

[10] Argonne National Lab, "Theta hpc traces," 2023, https://www.alcf.anl.gov/alcf-resources/theta (accessed August 2025).

[11] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and M. Taufer, "Scalable i/o-aware job scheduling for burst buffer enabled hpc clusters," in *ACM Int. Symp. on High-Performance Parallel and Distributed Computing*, 2016.