ORION: Integrated Runtime Modelling for Predicting Deep Learning Training Time

Alireza Pourali and Hamzeh Khazaei
York University
Department of Electrical Engineering and Computer Science
Toronto, Ontario, Canada
{alirezap,hkh}@yorku.ca

Abstract

Training deep learning models, especially Transformer-based and large-scale architectures, entails significant computational requirements and relies on precise coordination between accelerator execution and host-side data delivery. While prior performance prediction efforts have primarily focused on GPU compute modelling, the influence of the input pipeline, including CPU parallelism, data preprocessing, and storage I/O throughput, remains far less understood. This paper introduces ORION, an integrated runtime modelling framework that predicts iteration-level neural network training time by jointly characterizing GPU computation and host-side resource behaviour. ORION incorporates empirical measurements of data loading latency, CPU preprocessing scalability, and disk throughput into a unified analytical formulation that distinguishes compute-bound and I/O-bound execution regimes. By modelling the interaction between data ingestion and accelerator execution, ORION reveals critical host-induced bottlenecks and quantifies their contribution to end-to-end training latency. Across diverse vCPU counts, storage tiers, and neural network architectures, ORION achieves an average reduction of 44.36% in prediction error compared to the state-of-the-art GPU-centric baseline. Overall, ORION enables accurate, hardware-aware training time prediction and provides practical guidance for selecting balanced system configurations in modern deep learning environments.

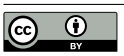
CCS Concepts

• **Computing methodologies** → **Machine learning**; • **General and reference** → *Performance*; • **Computer systems organization** → Cloud computing.

Keywords

Deep Learning Performance Modelling, CPU and Storage I/O Bottlenecks, Hardware-Aware Training Optimization

ACM Reference Format:
Alireza Pourali and Hamzeh Khazaei. 2026. ORION: Integrated Runtime Modelling for Predicting Deep Learning Training Time. In *Proceedings of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE '26), May 04–08, 2026, Florence, Italy*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3777884.3797001



ICPE '26, Florence, Italy

ACM ISBN 979-8-4007-2325-4/2026/05
https://doi.org/10.1145/3777884.3797001

1 Introduction

Deep neural networks (DNNs) have transformed modern machine learning, powering breakthroughs across computer vision and natural language processing [12, 14, 21, 37, 44]. Their rapid evolution has led to increasingly large architectures, with models now scaling to massive parameter counts [1, 3, 4, 8, 38, 40, 47]. While these advances have expanded model capability, they have also intensified the computational burden of training, resulting in longer runtimes and higher operational cost [15, 23, 24, 29, 31]. Consequently, training remains a major resource investment across both academic and industrial settings [6, 17, 25], making careful planning of hardware usage and workflow efficiency increasingly important [35].

Deep learning training relies on a continuous stream of batches prepared on the host before they reach the accelerator, yet this upstream path is far more complex than it appears. Raw samples must be fetched from storage, decoded, and transformed via augmentations, such as decompression, cropping, or geometric perturbations, standard in vision workloads [46]. These operations run on the CPU and are intended to overlap with GPU computation, but their effectiveness depends heavily on host-side capabilities [26]. In many workloads, limited vCPU parallelism or insufficient storage bandwidth slows data retrieval and preprocessing, causing the accelerator to wait idly. These data stalls, arising from delays in I/O or preprocessing, illustrate that training speed is shaped not only by the GPU but also by the host's ability to sustain a reliable flow of ready batches. This shifts the performance question from purely "Which GPU should be used?" to a broader system-level inquiry: *How do host-side resources such as vCPUs and storage tiers shape the overall pace and cost of deep learning training?*

Prior work on performance prediction has primarily focused on modelling computational demand or GPU execution characteristics [10, 30, 33, 45, 48, 49]. These approaches often rely on quantities such as floating-point operations or hardware counters [16] to estimate iteration time. While effective in compute-intensive regimes, these techniques provide limited visibility into scenarios where training is constrained by data ingestion rather than GPU throughput. In practice, end-to-end training speed emerges from the interplay between GPU computation and a host-driven data pipeline encompassing disk access, decoding, and CPU-side batch preparation. Without modelling these host side effects, predictions risk overlooking a substantial portion of real-world training variance.

In this work, we focus on *host-side* performance factors such as CPU parallelism and storage I/O throughput and their impact on end-to-end deep learning training time. While prior work has shown that data pipelines can substantially bottleneck end-to-end

training performance—whether due to redundant per-job data loading in shared clusters [19], input-pipeline stalls during preprocessing and fetching [26], or insufficient storage throughput in distributed settings [32], there remains a lack of systematic models that *quantify* how host-side factors such as vCPU provisioning and storage type (SSD vs. NVMe) contribute to per-batch latency. This gap becomes increasingly important as practitioners scale batch sizes or deploy multi-GPU accelerators, where even modest host under-provisioning can leave high-end devices underutilized.

In this paper, we introduce ORION, a framework that explicitly models the interaction between data preparation and GPU execution during neural network training. ORION integrates empirical measurements of per-batch data-loading latency, disk throughput, and CPU preprocessing scalability into a unified analytical formulation that distinguishes compute-bound and I/O-bound execution regimes. At its core, ORION represents the training step time as the combination of (i) the GPU's iteration cost and (ii) a host-side latency term that captures CPU-based preprocessing and disk-to-memory transfer effects under varying vCPU and storage configurations. By coupling this modelling approach with systematic variations in vCPU counts and storage tiers, ORION can attribute slowdowns to specific host components and predict how host reconfiguration, such as upgrading from SSD to NVMe or increasing available vCPUs, impacts per-batch training time.

Our experimental evaluation spans a comprehensive range of architectures, from lightweight token-mixing MLPs through convolutional networks and up to Transformer-based vision and language models, all assessed under diverse datasets and controlled vCPU and storage configurations. By revealing these host-induced inefficiencies, ORION enables hardware-aware performance prediction and provides actionable guidance for cost-efficient resource allocation. In particular, it helps practitioners answer questions such as: *How many vCPUs are sufficient before returns diminish? When does upgrading from SSD to NVMe materially reduce training time? Under which workloads does storage throughput become the dominant bottleneck?* Addressing these questions is essential for principled performance reasoning in modern deep learning workflows that span accelerators, CPUs, and storage systems. Across all evaluated architectures, ORION achieves an average reduction of 44.36% in prediction error compared to the state-of-the-art GPU-centric baseline, underscoring the importance of accounting for training-host effects, such as vCPU provisioning and storage throughput. Specifically, in this paper, we investigate the following research questions:

RQ1: How do training host factors, specifically vCPU parallelism and storage throughput, influence per-batch training time across widely used deep learning architectures, from MLPs to Transformer-based models? Training pipelines rely on CPU preprocessing and disk-to-memory transfers on the training host to supply data to the GPU. However, the extent to which vCPU counts and storage tiers (SSD vs. NVMe) on the training host affect end-to-end step time remains poorly understood. This question examines how these host-side resources shape performance across different model families and whether specific architectures are more sensitive to I/O or CPU bottlenecks.

RQ2: Can a unified analytical model accurately characterize compute-bound versus I/O-bound regimes by combining empirical GPU measurements with CPU scaling behaviour and storage I/O latency? GPU compute alone does not determine iteration speed: the overall step time depends on how quickly the host can prepare and deliver batches. This question evaluates whether incorporating CPU scaling dynamics and disk throughput into a single formulation allows ORION to reliably distinguish when the GPU is the bottleneck versus when host-side data preparation dominates.

RQ3: To what extent can host-aware performance modelling improve the predictability of training time for unseen model configurations and hardware setups? Because modern research uses diverse architectures and cloud instances, predicting training time for new configurations is essential for planning and cost control. This question investigates whether ORION's integration of vCPU and storage effects enables accurate generalization to unseen models (ranging from MLPs to Transformers) and to host configurations not encountered during training, thereby supporting robust, hardware-aware performance forecasting.

2 Related Work

Predicting the training time of deep neural networks (DNNs) has been explored from several angles, ranging from analytical layer-level modelling to hardware-aware learned predictors. Early work by Justus et al. [18] demonstrated that per-layer execution time can be estimated from structural parameters such as FLOPs and kernel shapes, enabling coarse prediction of per-batch runtime. While influential, their approach normalized architectural differences and did not incorporate the distinct computational patterns of Transformer-based models or modern accelerator stacks. Subsequent efforts advanced beyond analytical assumptions by relying on detailed hardware traces and execution profiling. Systems such as Habitat [11] capture the repetitive nature of training execution and use runtime measurements to forecast iteration-level performance across GPU types, whereas DNNPerf [10] models a neural network as a directed acyclic graph and applies graph neural networks to infer execution time from both operator-level and structural features. These profiling-based methods achieve high accuracy but require substantial model execution, limiting their utility for early-stage design and cross-architecture generalization.

A parallel line of research has highlighted the central role of the input data pipeline in determining end-to-end training speed. OneAccess [19] shows that conventional per-job data loading introduces severe redundancy in multi-tenant clusters, with hyperparameter tuning workloads wasting up to 97% of preprocessing computation. DS-Analyzer [26] presents the first comprehensive breakdown of data stalls in production systems, demonstrating that CPU-side preprocessing and data fetching frequently dominate total training time. Complementary to this, Hoard [32] uses a distributed NVMe-backed caching layer to mitigate storage bottlenecks and prevent GPU underutilization. These works reveal that host-side characteristics, including vCPU availability, data loader parallelism, and storage bandwidth, can sharply affect per-iteration time. Yet, they stop short of offering predictive models capable of quantifying these effects across architectures and hardware configurations.

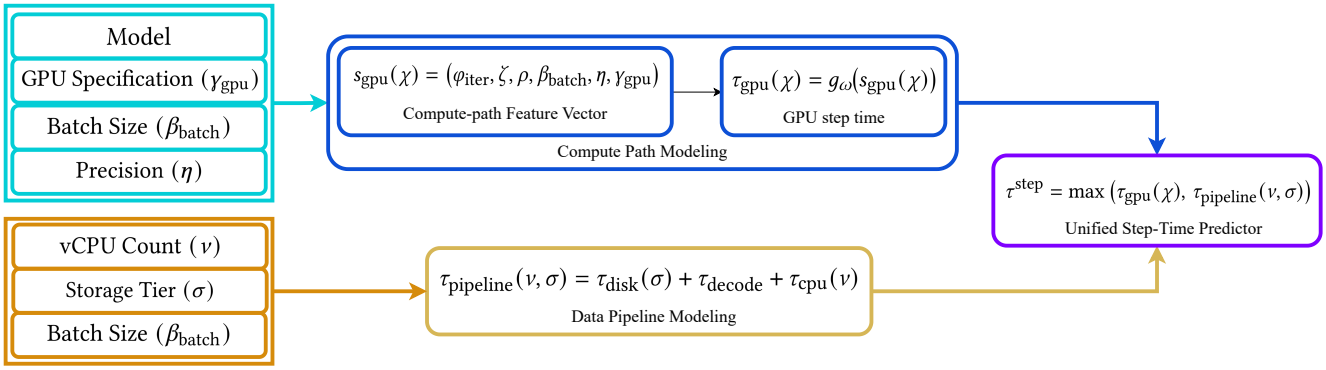


Figure 1: High Level Workflow of the ORION Training Time Prediction Framework

Learning-based approaches aim to generalize beyond handcrafted analytical models and trace-driven simulators. PreNeT [33] introduced a data-driven methodology that predicts iteration-level runtime from computational features such as FLOPs combined with GPU specifications. PreNeT [33] demonstrated strong generalization across architectures, but its predictions focused primarily on GPU compute and layer-level characteristics and did not account for host-induced delays such as insufficient vCPU provisioning or slow storage.

ORION builds on these foundations by extending the PreNeT [33] philosophy from layer-centric modelling toward full model-level, per-epoch training time prediction while explicitly integrating host I/O behaviour into the performance model. By systematically varying vCPU counts and storage types (SSD vs. NVMe), and by collecting training traces from models spanning MLPs, CNNs, and Transformers, our framework captures both compute and data-movement contributions to runtime. This shift yields the first predictive system capable of jointly reasoning about architectural structure and host-side data pipeline effects, filling a critical gap left by prior GPU-focused or profiling-dependent approaches.

3 ORION Design

The overall workflow of ORION is shown in Figure 1. At a high level, ORION converts a training configuration comprising a model architecture, GPU characteristics, vCPU provisioning, and storage tier into an analytical representation of its computational behaviour and its data delivery performance. These correspond to two tightly connected paths in the training process: a *compute path*, which determines the cost of executing a forward–backward optimizer step on the GPU, and a *data pipeline path*, which determines how quickly batches can be prepared and delivered from storage through CPU preprocessing.

This section provides a detailed account of the analytical foundations of ORION. We first introduce the architectural and execution descriptors used throughout the framework (3.1). We then describe the compute path modelling (3.2), followed by the data pipeline modelling (3.3). Finally, we show how ORION integrates these analytical components into a unified feature vector that supports accurate per-step time prediction (3.4).

Table 1: Architectural and execution parameters used by ORION. These symbolic quantities parameterize FLOPs formulations, memory surrogates, and data-pipeline latency models.

Parameter	Description
Depth (K)	Number of repeated blocks or layers (e.g., convolutional blocks, transformer layers).
Width / Embedding (Δ)	Hidden or embedding dimension of tokens or feature channels.
Token Count (P)	Number of spatial tokens or patches in token-mixing MLPs.
Sequence Length (T)	Number of tokens or time steps (transformers).
Attention Heads (H)	Number of self-attention heads in transformer blocks.
FFN Width (d_{ff})	Expansion dimension of the feed-forward network.
Parameter Count (Π)	Total number of trainable parameters.
Activation Size (Λ)	Activation memory footprint per forward pass.
Batch Size (β_{batch})	Number of samples processed per iteration.
Precision (η)	Numerical precision (FP16 or FP32).
GPU Specification (γ_{gpu})	GPU hardware configuration (e.g., Memory bandwidth, clock rates).
vCPU Count (ν)	Number of CPU workers available for preprocessing.
Storage Tier (σ)	Storage type used for dataset access (SSD or NVMe).

3.1 Architectural and Execution Parameters

ORION does not rely on a layer-by-layer graph representation. Instead, it builds on symbolic *architectural parameters* that provide sufficient structure for deriving precise FLOPs expressions for each supported model family: Token-mixing MLPs, CNNs and Transformers. These parameters capture depth, dimensionality, kernel structure, token/sequence topology, and gating patterns that determine computational cost. In addition, ORION records *execution parameters* that influence runtime but not FLOP structure, such as batch size, precision, GPU model, vCPU count, and storage tier.

Table 1 consolidates all parameters used throughout the analytical derivations and the final feature vector.

3.2 Compute Path Modelling

The first analytical component of ORION models the execution behaviour of the GPU. Based on the architectural parameters in Table 1, ORION computes a *FLOPs surrogate* for an entire forward–backward optimizer iteration and augments it with memory-based features that characterize computational intensity.

Forward-Pass FLOP Derivations. The forward pass FLOP count φ_{fwd} is obtained from architecture-specific closed-form expressions derived directly from the structural parameters of each supported model family. These analytical expressions capture the exact tensor algebra underlying convolutional, attention, token-mixing operators, yielding a high-fidelity characterization of computational workload. This design is inspired by the symbolic FLOP modelling philosophy introduced in PreNeT [33]; however, as discussed in Section 3.1, ORION adopts a fundamentally different approach. Rather than decomposing the network into a sequence of per-layer timing predictors, ORION aggregates the architectural parameters into a unified, iteration-level analytical representation. This shift enables the framework to model the compute cost without relying on layer granular profiling while still preserving hardware-agnostic generalization across heterogeneous model families and accelerator platforms.

Token-Mixing MLPs. For architectures such as MLP-Mixer [41], ResMLP [42], or AS-MLP [22], with token count P, embedding dimension Δ, expansion ratios r_t and r_c in token-mixing and channel-mixing MLPs, and K blocks, we define the forward-pass FLOPs using standard multiply accumulate accounting, where the factor 4 accounts for the two linear projections (token-mixing and channel-mixing), each involving one multiplication and one addition per operation:

$$\varphi_{\text{fwd}}^{\text{mlp}} = K \cdot 4P\Delta(r_t + r_c)$$

Convolutional Networks (CNNs) [28]. For a convolutional layer with input channels C_{in}, output channels C_{out}, group count G, kernel size $K_h \times K_w$, and output spatial size $H_{\text{out}} \times W_{\text{out}}$, we define the FLOPs using standard multiply accumulate accounting, where the factor 2 corresponds to one multiplication and one addition performed for each kernel application:

$$\varphi_\ell^{\text{conv}} = 2 \cdot \frac{C_{\text{in}}}{G} \cdot C_{\text{out}} \cdot K_h K_w \cdot H_{\text{out}} W_{\text{out}}$$

Fully-connected layers contribute:

$$\varphi_\ell^{\text{fc}} = 2 d_{\text{in}} d_{\text{out}}$$

Summing across all layers yields:

$$\varphi_{\text{fwd}}^{\text{cnn}} = \sum_{\ell \in \text{Conv}} \varphi_\ell^{\text{conv}} + \sum_{\ell \in \text{FC}} \varphi_\ell^{\text{fc}}$$

Transformers [44]. With sequence length T, embedding dimension Δ, number of heads H (with head dimension $d_h = \Delta/H$), feed-forward width d_{ff}, and K blocks, the forward FLOPs are:

$$\varphi_{\text{fwd}}^{\text{tr}} = K\left(4T\Delta^2 + 2HT^2 d_h + 4T\Delta d_{\text{ff}}\right)$$

Iteration-Level Compute Surrogate. ORION expands the forward FLOPs into a full iteration cost:

$$\varphi_{\text{iter}} = \kappa \cdot \varphi_{\text{fwd}},$$

where $\kappa \in [2.5, 3.5]$ models the additional computation incurred by backpropagation and optimizer updates. The lower end of this interval reflects the theoretical structure of modern deep learning workloads, where the backward pass typically introduces roughly twice the computational load of the forward pass due to gradient propagation through every operator. The upper end accounts for practical sources of overhead that arise during training, including optimizer state updates (e.g., Adam/AdamW), mixed-precision bookkeeping, kernel fusion interactions, and memory-access imbalance across layers.

By adopting this bounded multiplier, ORION captures realistic iteration-level computational cost while avoiding the need for explicit layer-by-layer accumulation. This enables the learned predictor to absorb implementation-specific effects, such as fused kernels, precision modes, and GPU scheduling—while maintaining a concise and consistent analytical representation of the full training iteration.

Memory Surrogate and Arithmetic Intensity. The memory surrogate is defined as:

$$\zeta = \Pi_{\text{bytes}} + \Omega_{\text{bytes}} + \Lambda_{\text{bytes}},$$

where Π_{bytes} denotes parameter storage, $\Omega_{\text{bytes}} = m_{\text{opt}}\Pi_{\text{bytes}}$ denotes optimizer state (with $m_{\text{opt}} = 2$ for Adam/AdamW and $m_{\text{opt}} = 1$ for SGD), and Λ_{bytes} denotes activation memory. The arithmetic density is defined as:

$$\rho = \varphi_{\text{iter}}/\zeta,$$

and indicates whether a workload is compute-intensive or memory-limited.

All compute-path analytical quantities are then aggregated into a compute-path feature vector:

$$s_{\text{gpu}}(\chi) = \left(\varphi_{\text{iter}}, \zeta, \rho, \beta_{\text{batch}}, \eta, \gamma_{\text{gpu}}\right),$$

where χ denotes the full training configuration comprising both architectural parameters and execution settings. Each component captures a distinct facet of the iteration-level compute workload: φ_{iter} encodes total per-iteration FLOPs, ζ represents the parameter, optimizer-state, and activation memory footprint, $\rho = \varphi_{\text{iter}}/\zeta$ denotes arithmetic intensity, β_{batch} is the batch size, η specifies numerical precision, and γ_{gpu} summarizes accelerator hardware characteristics (e.g., SM count, bandwidth, clock rates).

ORION derives the GPU step time via a learned regressor g_ω:

$$\tau_{\text{gpu}}(\chi) = g_\omega\big(s_{\text{gpu}}(\chi)\big).$$

This $\tau_{\text{gpu}}(\chi)$ term forms the compute-path contribution used in the unified step-time predictor.

3.3 Data Pipeline Modelling

The second analytical component of ORION models the latency introduced by the data ingestion and preprocessing pipeline. Before each batch reaches the GPU, it must be (i) read from storage, (ii) decoded, and (iii) transformed through CPU preprocessing (e.g.,

cropping, normalization, tokenization). These stages run concurrently with GPU execution, and the slower of the two determines the per-step time.

Data Pipeline Latency Formulation. The total pipeline latency is defined as:

$$\tau_{\text{pipeline}}(\nu, \sigma) = \tau_{\text{disk}}(\sigma) + \tau_{\text{decode}} + \tau_{\text{cpu}}(\nu),$$

where each term models a distinct stage in the host-side data ingestion path. Separating these components allows ORION to capture the individual contributions of storage bandwidth, data-format decoding, and CPU-side preprocessing. In particular, the decode term accounts for the cost of transforming compressed samples (e.g., JPEG, PNG, or tokenized text) into raw tensors before augmentation.

The three components are:

- **Disk transfer time $\tau_{\text{disk}}(\sigma)$:** time to fetch D_B bytes from storage tier σ.
- **Decode latency τ_{decode}:** time to convert encoded formats into tensors (image decode, text tokenize). This term is invariant to ν but depends on data type and compression.
- **CPU preprocessing time $\tau_{\text{cpu}}(\nu)$:** time required to apply augmentations and transformations with ν workers.

Storage Model. Let D_B be the bytes read per batch. Then:

$$\tau_{\text{disk}}(\sigma) \approx \frac{D_B}{B_{\text{disk}}(\sigma)},$$

where $B_{\text{disk}}(\sigma)$ is the sustained bandwidth of SSD or NVMe (typically 0.4–0.7 GB/s vs. 2–7 GB/s).

Decode Model. Decode latency is modelled as a per-sample cost:

$$\tau_{\text{decode}} = \beta_{\text{decode}} \cdot \beta_{\text{batch}},$$

where β_{decode} denotes the average decode time per sample, obtained once per dataset through a small calibration step. This linear model reflects the fact that image and text decoding typically proceed independently across samples and incur no meaningful inter-sample reuse, producing a batch-level decode cost that scales proportionally with the batch size.

CPU Preprocessing Model. To capture the effect of parallel preprocessing on the host, ORION adopts an inverse form of the Universal Scalability Law (USL) [13], a classical model used to characterize how parallel throughput degrades in the presence of resource contention and worker coherency overheads. Let $\tau_{\text{cpu}}(1)$ denote the single-worker preprocessing time. The latency with ν workers is expressed as:

$$\tau_{\text{cpu}}(\nu) = \frac{\tau_{\text{cpu}}(1)}{\nu \big/ (1 + \alpha(\nu - 1) + \beta\, \nu(\nu - 1))},$$

where the parameters α and β model the two dominant slowdown mechanisms identified by USL. The contention term α captures interference on shared CPU resources, such as the cache hierarchy, memory bandwidth, and Data Loader queueing, while the coherency term β accounts for higher-order interactions that emerge at larger worker counts, including kernel scheduling overheads, Non-Uniform Memory Access effects, and cross-core synchronization induced by augmentations or I/O buffering. This inverse USL formulation yields realistic scaling behaviour: near- linear speedup at small ν, diminishing gains as contention grows, and eventual latency increases when coherency penalties dominate. Together with the disk-transfer and decode terms, this model enables ORION to determine when host-side data availability becomes the dominant factor in per-step training time.

3.4 Unified Step-Time Predictor

ORION assembles its final analytical feature vector by combining compute-path features (FLOPs, memory, arithmetic intensity, precision, GPU specifications) with data-pipeline features (vCPU count, storage tier, decode cost, and preprocessing latency). The predicted per-step time is:

$$\tau^{\text{step}} = \max\left(\tau_{\text{gpu}}(\chi),\ \tau_{\text{pipeline}}(\nu, \sigma)\right)$$

This parallel execution formulation enables a single learned regressor to handle both compute-dominated and pipeline-dominated workloads, achieving generalization across diverse architectures and hardware configurations, including those unseen during training.

4 Evaluation and Discussion

ORION is evaluated across diverse GPU architectures, host configurations, and deep learning models to assess its ability to predict per batch training time under heterogeneous compute and data delivery conditions. Building on the analytical foundation of PreNeT [33], the only prior work with a comparable feature based formulation for training time prediction, ORION extends this approach by incorporating host side factors such as vCPU provisioning and storage tier and by using a unified model level FLOPs representation. Accordingly, we compare ORION exclusively against PreNeT, as other approaches target different granularities or rely on assumptions that are not directly comparable to ORION's host aware, iteration-level setting. All experiments were conducted on RunPod cloud instances provisioned with AMD EPYC 7542 processors, where we systematically varied the number of active vCPUs and ran models on both SSD and NVMe-backed instances to capture the full spectrum of data delivery behaviour. To ensure reproducibility, we release all code and additional results omitted due to page limits on our GitHub repository.[1]

4.1 GPU Coverage and Hardware Diversity

To ensure broad hardware coverage, we benchmark eight modern NVIDIA GPUs, spanning datacenter accelerators and high-performance workstation and consumer devices. These GPUs exhibit substantial variation in streaming multiprocessor counts, memory capacity, memory technology (GDDR6, GDDR7, HBM2e), memory bandwidth, and PCIe generation, resulting in diverse compute-memory balance and host-device interaction patterns.

Across all runs, we vary the effective vCPU count by adjusting the number of data-loading worker processes, and we train on both SSD and NVMe storage tiers. This produces training samples that span CPU-limited, storage-limited, and GPU-limited settings.

[1] https://github.com/pacslab/ORION

Table 2: GPU models and key specifications used in our experiments.

GPU Model	SM	Memory	BW (GB/s)	Bus
NVIDIA L4	60	24GB GDDR6	300	PCIe 4.0
NVIDIA H100 PCIe	114	80GB HBM2e	2040	PCIe 5.0
NVIDIA A100 80GB PCIe	108	80GB HBM2e	1940	PCIe 4.0
NVIDIA L40S	142	48GB GDDR6	864	PCIe 4.0
NVIDIA RTX 5090	170	32GB GDDR7	1790	PCIe 5.0
NVIDIA A40	84	48GB GDDR6	696	PCIe 4.0
NVIDIA RTX 4000 Ada	48	20GB GDDR6	360	PCIe 4.0
NVIDIA L40	142	48GB GDDR6	864	PCIe 4.0

4.2 Model and Dataset Coverage

We evaluate ORION on a broad collection of deep neural architectures spanning multiple families of modern networks. The benchmark suite includes token-mixing MLPs (MLP-Mixer [41], ResMLP [42], AS-MLP [22]), convolutional networks (ResNet-50 [12], VGG-16 [37], DenseNet-121 [14]), vision transformers (DeiT-Tiny [43], ViT [9]), encoder-style language models (DistilBERT [34]), and encoder–decoder Transformer [44] used in sequence-to-sequence tasks.

These architectures are trained on widely used datasets across both vision and language domains, including CIFAR-10, CIFAR-100 [20], STL-10 [5], and TinyImageNet [7] for image classification; SST2 [39] and IMDB for text classification; and WMT14 EN–DE [2] and CNN/DailyMail [27] for machine translation and summarization. This combination spans low and high-resolution inputs, short and long sequences, and diverse activation and memory footprints.

Each model–dataset pairing is executed across multiple hyperparameter settings, varying batch sizes, learning rates, optimizers, precision modes, and host configurations, to generate a comprehensive, heterogeneous set of training time measurements. This rich benchmark distribution ensures that ORION is evaluated across a broad spectrum of computational and data-access characteristics.

4.3 Parameter Configurations and Data Generation

To construct a comprehensive dataset, we evaluate a coordinated set of parameter configurations that vary both model-level and host-level factors across all architectures in our study. Batch sizes range from 4 to 256, and learning rates span from 5×10^{-4} to 3×10^{-3}. We examine several commonly used optimizers, including AdamW, Adam, SGD, and Adafactor, under both FP32 and mixed-precision modes. To capture host-side variability, we vary the number of active vCPUs (4, 8, and 16) and evaluate performance under both SSD and NVMe storage tiers.

Architectural hyperparameters are also systematically varied to reflect the diversity of the benchmarked models. These include convolutional kernel choices, hidden-layer sizes, embedding dimensions, model depth, number of attention heads, feed-forward multipliers, vocabulary sizes, and sequence lengths. This ensures the dataset spans a broad range of computational intensities, memory footprints, and data-access patterns.

Each configuration is profiled for several warmup iterations followed by 25 measured batches, whose per-step timings are averaged to obtain stable estimates. For every run, we record the time spent in GPU computation and CPU–I/O data loading, as well as the fraction of the step dominated by the data pipeline. The resulting dataset covers hundreds of thousands of configuration points across GPUs, models, vCPU allocations, and storage types, forming the foundation for the predictive modelling presented in the following sections.

4.4 Regression Model and Evaluation Protocol

We evaluate several supervised learning models, including linear predictors, tree-based ensembles, and neural regressors, on the engineered feature sets derived from the collected data. Consistent with the PreNeT [33] baseline, a gradient-boosted decision tree model provided the most stable and accurate predictions across architecture families and hardware settings. ORION therefore uses a boosted ensemble configured with a large number of trees, moderate depth, conservative shrinkage, and subsampling across both samples and features. This enables the model to capture nonlinear interactions among computational characteristics, GPU specifications, and host-level settings.

Our primary evaluation protocol uses cross-fold evaluation. For each architecture family, we construct a stratified five-fold split across all configurations, ensuring that each fold contains a diverse combination of batch sizes, learning rates, GPUs, vCPU settings, and storage tiers. The regressor is trained on four folds and evaluated on the remaining fold, yielding per-sample predictions without overlap between training and testing.

We also conduct a leave-one-GPU-out evaluation, in which data from one GPU is excluded from training and used only for testing. This examines ORION's ability to generalize to unseen hardware using only analytical features and high-level specifications. The results of this experiment are reported in the next section.

4.5 Evaluation Metric (RMSE)

We use the Root Mean Square Error (RMSE) as the primary evaluation metric. RMSE penalizes large deviations more strongly than absolute-error metrics, which is essential when occasional slow batches disproportionately affect end-to-end training behaviour. For a set of N samples with predicted step times $\hat{y}_i$ and measured times y_i, RMSE is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}.$$

We compute RMSE for both the PreNeT [33] GPU-centric baseline and for ORION, enabling a direct comparison of prediction quality. This metric is applied in both cross-fold evaluation and in the leave-one-GPU-out setting, allowing us to measure how well each model generalizes across host configurations, storage tiers, and entirely unseen hardware. Inspired by prior work such as PreNeT [33], RMSE serves as a consistent and interpretable measure of predictive accuracy across the diverse architectures and hardware platforms used in our study.

Table 3: Cross-fold RMSE (ms) across all model families, along with the average measured step time (ms) for contextualization. ORION significantly outperforms the PreNeT [33] baseline, with gains up to 92.87%.

Model	PreNeT [33]	ORION	Improvement	Avg. Step
AS-MLP [22]	72.25	5.15	92.87%	69.92
MLP-Mixer [41]	69.90	5.62	91.97%	56.72
ResMLP [42]	5.23	3.61	30.93%	25.40
DenseNet-121 [14]	10.34	6.36	38.43%	59.86
ResNet-50 [12]	5.09	3.58	29.66%	31.05
VGG-16 [37]	1.10	0.78	28.91%	47.27
DeiT-Tiny [43]	9.15	4.12	54.95%	42.58
DistilBERT [34]	4.46	3.99	10.46%	22.50
ViT [9]	4.41	2.99	32.18%	58.83
Transformer [44]	13.63	9.10	33.26%	151.77
Average	19.56	4.53	44.36%	–

Table 4: Effect of vCPU provisioning on prediction accuracy (ORION). RMSE remains consistent across 4, 8, and 16 vCPUs, indicating limited sensitivity to CPU parallelism.

Model	4 vCPUs	8 vCPUs	16 vCPUs
AS-MLP	3.77	3.93	5.58
MLP-Mixer	3.67	3.95	6.08
ResMLP	3.65	3.25	3.26
DeiT-Tiny	3.59	3.64	5.96
DistilBERT	3.81	3.82	3.91
ViT	3.23	3.49	3.68
Transformer	8.35	9.80	9.37
ResNet-50	3.51	3.24	3.66
DenseNet-121	5.86	6.06	6.31
VGG-16	0.69	0.66	0.59

5 Results

This section evaluates ORION under the cross-fold and leave-one-GPU-out protocols described earlier. Our analysis focuses on three questions: (i) how ORION compares to the state-of-the-art baseline, (ii) how host-level configuration (vCPU and storage tier) affects prediction accuracy, and (iii) how these effects vary across different model families.

5.1 Overall Cross-Fold RMSE

Table 3 summarizes the end-to-end RMSE (ms) across all major architectures. ORION consistently outperforms the GPU-centric PreNeT [33] baseline, achieving improvements ranging from 10.46% (DistilBERT) to 92.87% (AS-MLP). These gains confirm that GPU compute time alone is insufficient for accurately predicting full training step latency, especially on host-sensitive architectures.

Across all models, ORION achieves substantial reductions in error. The most significant benefits occur in token-mixing MLP architectures such as AS-MLP and MLP-Mixer, whose lightweight compute footprints make them highly sensitive to host-level bottlenecks. In these cases, the GPU-centric baseline severely underestimates the cost of CPU-side decoding, batch preparation, and storage-to-memory transfers, resulting in RMSE values exceeding 69 ms. By contrast, ORION incorporates host-level predictors (vCPU parallelism, storage throughput, memory-bandwidth proxies and pipeline stall ratios), enabling accurate modelling of I/O-bound behaviour.

Architectures with heavier compute workloads, such as convolutional networks and Transformer models, exhibit smaller gaps between GPU compute time and full step time. Even in these compute-dominated settings, ORION improves RMSE by 10–55%, in large part because it relies on model-level FLOPs rather than layer-wise profiling of PreNeT [33] and therefore captures the overall computational demand while simultaneously modelling data delivery behaviour.

5.2 Impact of vCPU Provisioning

To understand the sensitivity of ORION's predictions to host-side CPU parallelism, we evaluate RMSE as the number of available vCPUs varies from 4 to 16. Table 4 reports ORION's cross-fold RMSE for each model under these three vCPU configurations.

Overall, ORION's RMSE varies only modestly across the three vCPU levels. For most architectures, the differences between 4, 8, and 16 vCPUs remain within a narrow range, suggesting that CPU parallelism is not a dominant factor influencing prediction accuracy. This result aligns with our empirical observations of the data pipeline: per-step latency is often shaped more by storage bandwidth, dataset decode cost, and memory-access patterns than by the degree of CPU parallelism. While increasing vCPUs can marginally improve input throughput, especially for augmentation-heavy workloads, its effect on end-to-end iteration time is limited because many preprocessing stages, particularly decode and memory fetch, remain serial or I/O-bound. Consequently, ORION maintains stable accuracy across vCPU settings, highlighting that storage tier and dataset characteristics exert a more decisive influence on host-side bottlenecks than raw CPU core counts.

5.3 Impact of Storage Tier

In contrast to vCPUs, the storage tier has a clear and consistent impact on prediction accuracy, especially for the GPU-only baseline. Table 5 reports mean RMSE (in ms) for SSD and NVMe, averaged over vCPU settings (4, 8, 16) for every architecture in our evaluation suite.

For PreNeT [33], moving from SSD to NVMe reduces RMSE for nearly all models, with improvements ranging from moderate to substantial. Token-mixing MLPs, compact models, and lightweight CNNs exhibit the greatest reductions. For example, PreNeT [33] error decreases from 6.33 ms to 3.57 ms for ResMLP, from 11.53 ms to 5.80 ms for DeiT-Tiny, and from 1.25 ms to 0.88 ms for VGG-16. Larger reductions are also visible in the Transformer family, including DistilBERT and the full Transformer, where SSD-induced stalls widen the gap between GPU compute and end-to-end step time.

For ORION, the effect of the storage tier is more nuanced. Because ORION explicitly models data-loading delays, its RMSE is consistently low for both SSD and NVMe. NVMe provides modest improvements for many architectures such as AS-MLP, MLP-Mixer, ResMLP, VGG-16, DenseNet-121, DistilBERT, and ResNet-50.

Table 5: Effect of storage tier on prediction accuracy. Values are mean RMSE (ms) across vCPU configurations (4, 8, 16).

Model	PreNeT [33] (SSD)	ORION (SSD)	PreNeT [33] (NVMe)	ORION (NVMe)
AS-MLP	68.15	5.65	61.21	4.43
MLP-Mixer	66.68	6.32	58.98	4.57
ResMLP	6.33	3.82	3.57	3.39
DeiT-Tiny	11.53	3.66	5.80	4.40
ViT	4.36	2.42	4.45	3.47
ResNet-50	5.20	3.69	4.96	3.47
DenseNet-121	10.64	6.63	10.02	6.08
VGG-16	1.25	0.87	0.88	0.65
DistilBERT	4.56	4.13	4.35	3.85
Transformer	14.07	8.98	13.17	9.17

Some models, including DeiT-Tiny, ViT, and the full Transformer, show similar or slightly higher RMSE under NVMe, reflecting residual variability dominated by GPU execution rather than storage throughput.

Overall, these results confirm that storage throughput, rather than CPU parallelism, is the dominant host-side factor shaping end-to-end training-time predictability. PreNeT's GPU-only formulation suffers on SSD-bound workloads, whereas ORION remains robust across both tiers, with NVMe particularly beneficial for the most I/O-sensitive architectures.

5.4 Architectural Sensitivity

Storage sensitivity of token-mixing MLPs and lightweight models. Architectures such as AS-MLP, MLP-Mixer, ResMLP, and compact models like VGG-16 show the strongest dependence on the storage tier. Their relatively low computational intensity makes them highly susceptible to SSD-bound stalls, resulting in large PreNeT [33] errors that shrink significantly under NVMe. ORION maintains low error for all of these architectures, with NVMe providing further improvements.

Moderate storage effects in CNNs and Transformers. Models including ResNet-50, DenseNet-121, ViT, DistilBERT, and the full Transformer exhibit moderate reductions in PreNeT [33] error when using NVMe. These models involve more GPU computation but remain partially constrained by data availability. ORION remains accurate across both tiers, with NVMe improving accuracy for many of these models and producing similar or slightly higher RMSE for DeiT-Tiny, ViT, and the Transformer due to GPU-side variability.

Minimal impact of vCPU scaling. Across all architectures, increasing vCPUs from 4 to 16 changes RMSE by less than 1 ms. This behaviour reflects the dominance of single-threaded dataset decode, memory access patterns, and storage throughput, which limits the influence of CPU parallelism on prediction accuracy.

Greater benefits for I/O-heavy and low-compute architectures. Models whose step time is largely governed by input processing, such as token-mixing MLPs, VGG-16 and compact Transformers benefit the most from both faster storage and host-aware modelling. PreNeT [33] consistently overestimates training time on SSD-bound workloads, whereas ORION effectively models storage-induced delays and maintains low error across all tiers.

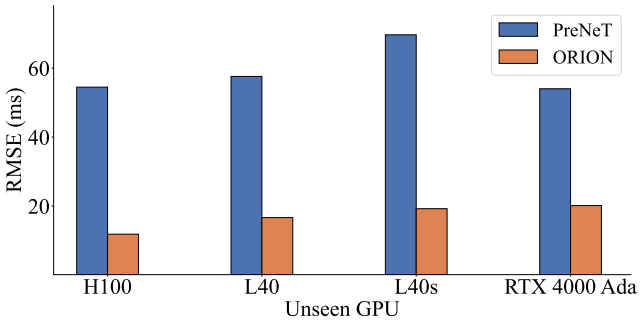


Figure 2: Leave-One-GPU-Out RMSE for MLP models. Unseen GPUs: H100, L40, L40S, RTX 4000 Ada. ORION reduces RMSE by large margins across all GPUs, reflecting the strong host-sensitivity of token-mixing MLPs.

5.5 Unseen-GPU Generalization (LOGO Evaluation)

To evaluate how well ORION generalizes to hardware configurations never seen during training, we apply a Leave-One-GPU-Out (LOGO) protocol across the MLP, CNN, and Transformer families. For each architecture group, models are trained on data from all but one GPU, and the held-out GPU is used exclusively for testing. This setup reflects realistic deployment scenarios where performance must be predicted on new accelerators without prior measurements. Due to page constraints, we report results for a representative subset of unseen GPUs per architecture family, selected to span different architectural generations and memory bandwidth regimes. Results on the remaining GPUs follow the same trends and are omitted for brevity.

Figures 2–4 summarize RMSE achieved by ORION and PreNeT. Across all families and all unseen GPUs, ORION consistently achieves *lower* RMSE compared to the PreNeT [33] baseline. This gap is particularly pronounced for lightweight and I/O-sensitive models such as token-mixing MLPs and compact CNNs. These models have

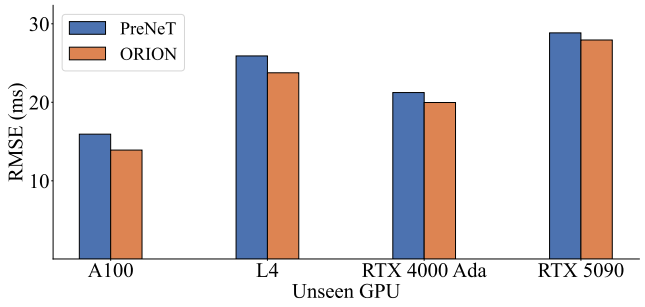


Figure 3: LOGO RMSE for CNN models on unseen GPUs (A100, RTX 4000 Ada, L4, RTX 5090). ORION consistently outperforms the GPU-only baseline, especially on lightweight CNNs where storage and CPU bottlenecks dominate.

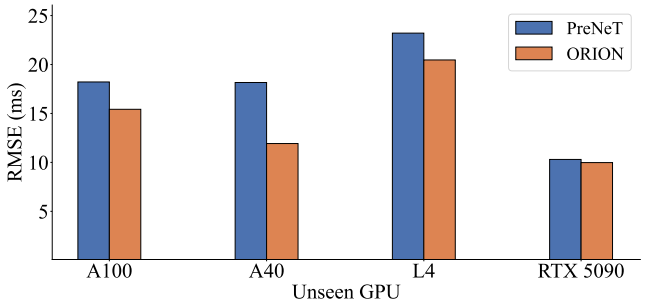


Figure 4: LOGO RMSE for Transformer models on unseen GPUs (RTX 5090, A40, A100, L4). Even in compute-heavy architectures, ORION achieves 20–50% lower RMSE, demonstrating robustness to architectural variation across GPU families.

small GPU compute footprints, making their end-to-end training-step time dominated by host-side factors (decode overhead, CPU parallelism, storage throughput). Because PreNeT [33] is trained only on GPU compute time (T_{gpu}), it systematically *underestimates* the step time on unseen accelerators, an effect amplified when the unseen GPU differs architecturally from the GPUs in the training set.

In contrast, ORION explicitly models CPU preprocessing, disk-to-memory transfer costs, and GPU–host interactions. As a result, its predictions remain stable even when the held-out GPU belongs to a different family (e.g., Hopper vs. Ada vs. Ampere). This robustness is most visible in the MLP LOGO evaluation (Fig. 2), where PreNeT [33] exhibits RMSE exceeding 50–70 ms on H100, L40, and L40S, while ORION maintains RMSEs from 11.85 ms to 20.14 ms. CNNs and Transformers also show consistent reductions: ORION improves RMSE by 20–60% across all unseen GPUs, even when GPU compute dominates overall cost (e.g., A100 or RTX 5090). These results demonstrate that incorporating host-level features is essential for robust cross-hardware prediction.

Overall, the LOGO experiments confirm that:

(1) **ORION generalizes effectively to unseen GPUs,** even when GPU architecture, memory hierarchy, or bandwidth characteristics change.
(2) **Host-side modelling is essential,** especially for MLPs and lightweight CNNs where storage and CPU throughput dominate latency.
(3) **GPU-only predictors such as PreNeT [33] systematically fail** on new hardware because they ignore CPU and storage effects, leading to large underestimation errors.

These findings highlight the necessity of unified compute and I/O modelling for predicting end-to-end training time across diverse hardware platforms.

5.6 Generalization Across Batch Sizes

In addition to cross-GPU robustness, an accurate training-time predictor must generalize across *different batch sizes*, which significantly alter both GPU compute demand and host-side memory pressure. To isolate this effect, we evaluate ORION on GPUs that were *not* used as unseen devices in the previous subsection. Accordingly, MLP, CNN, and Transformer models are evaluated on A100, L40, and L40S, respectively, each excluded from that family's LOGO evaluation presented in the paper.

Figures 5–7 report measured and ORION-predicted iteration times across small, medium, and large batch sizes for representative models in each family. The percentage differences between actual and predicted times are shown above each pair of bars to illustrate the magnitude of the prediction error. Across all settings, ORION tracks the growth in iteration time with high fidelity, consistently capturing both the *rate* and *magnitude* of latency increase. This behaviour highlights ORION's ability to model how batch scaling simultaneously stresses GPU compute, memory bandwidth, and host-side preprocessing.

MLP models. Figure 5 shows AS-MLP, MLP-Mixer, and ResMLP on the A100. Even though these models are susceptible to CPU decode and storage throughput, ORION closely matches the measured increase in latency as batch size grows from 32 to 256. In all cases, prediction error remains low, with the largest deviation under 25%. ORION successfully captures transitions from moderately I/O-bound to compute-dominated regimes as the batch expands.

CNN models. Figure 6 presents results for DenseNet121, ResNet50, and VGG16 on the L40. These models exhibit a smoother scaling curve driven largely by GPU compute intensity. ORION accurately models this trend for all three architectures, maintaining tight alignment with measured values across batch sizes 64, 128, and 256. The prediction error remains below 18% for most configurations, demonstrating ORION's robustness even on hardware not used in the unseen-GPU experiments.

Transformer models. Figure 7 shows DeiT-Tiny, DistilBERT, ViT, and the full Transformer architecture on the L40S. Transformers combine nontrivial data loading overheads with large per-token compute, making batch-size scaling particularly challenging to predict. Nevertheless, ORION closely follows the measured latency curves from small to medium to large batch sizes. Errors remain between 1–30% depending on model size, and ORION captures both

the steep scaling behaviour of DistilBERT and the more compute-linear profile of ViT and Transformer.

Together, these results show that ORION not only generalizes across unseen GPUs, but also across varying batch sizes that significantly alter the compute I/O balance. Unlike PreNeT [33], which fails to account for increases in host-side preparation cost or memory bandwidth demand, ORION incorporates batch-aware features (e.g., activation bytes, roofline interactions, memory pressure) that allow it to adapt its predictions as batch size grows. This makes ORION suitable for real-world scheduling and resource planning where batch size is frequently tuned.

6 Discussion

Our results demonstrate that ORION substantially improves modelling of end-to-end training behaviour by explicitly capturing the interaction between accelerator execution and the host-side input pipeline. Unlike GPU-centric predictors such as PreNeT [33], which implicitly assume that iteration time is dominated by computation, ORION treats storage throughput, decode cost, and CPU preprocessing latency as first-class contributors. This design enables it to distinguish compute-bound from pipeline-bound regimes and to attribute latency variations to the dominant system bottleneck.

A key insight from our evaluation is that data ingestion governs training throughput for a wide class of models with lightweight compute footprints, particularly token-mixing MLPs. For these architectures, changes in vCPU parallelism, storage tier, or batch size can significantly alter step time in ways that are invisible to GPU-only predictors. ORION accurately captures such effects through unified modelling of disk bandwidth, decode cost, and CPU scaling, but its predictions assume that data pipelines exhibit stable throughput characteristics and do not undergo abrupt contention or interference from external workloads.

For CNNs and Transformer-based models, host-side effects are more moderate but remain non-negligible. While heavier GPU computation reduces pipeline-induced variance, ORION continues to identify the correct bottleneck regime across batch sizes, storage tiers, and unseen GPUs. Nevertheless, prediction accuracy may degrade in scenarios where system conditions change dynamically (e.g., transient I/O congestion, non-deterministic data augmentation, or asynchronous pipeline reconfiguration), as such effects are not explicitly modelled. Overall, these results highlight that accurate training-time prediction relies on balanced assumptions about compute, storage, and data delivery stability rather than GPU characteristics alone.

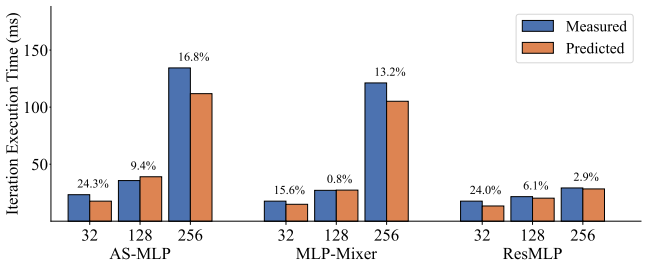


Figure 5: ORION-predicted iteration time for MLP models across batch sizes on the A100 GPU. ORION tracks the increase in latency accurately across small, medium, and large batches.

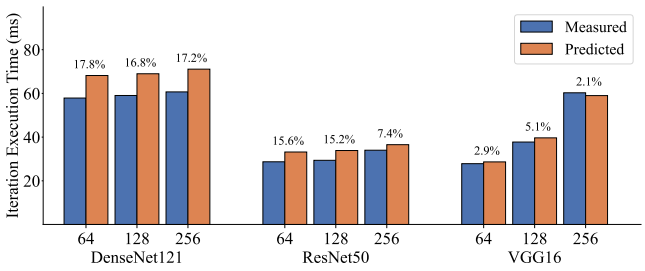


Figure 6: Batch-size scaling for CNN models (DenseNet121, ResNet50, VGG16) on the L40 GPU. ORION remains accurate as the batch size grows from 64 to 256, capturing both compute and memory–bandwidth effects.

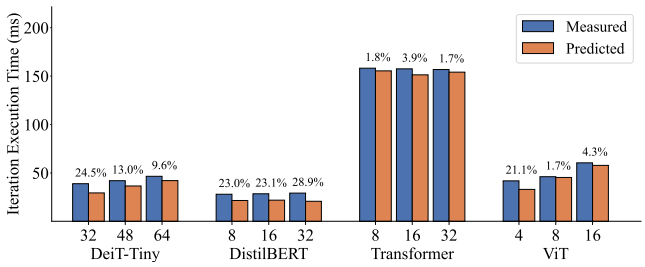


Figure 7: Batch-size generalization for Transformer models on the L40S GPU. ORION effectively follows the measured latency curve for DeiT-Tiny, DistilBERT, ViT, and Transformer, even when scaling behaviour varies across models families.

7 Conclusion and Future Work

In this paper, we introduced ORION, a unified framework for predicting iteration-level deep learning training time by jointly modelling GPU computation and host-side data delivery. By integrating analytical FLOPs surrogates with empirical models of disk throughput, decode latency, and CPU preprocessing scalability, ORION provides accurate predictions across a broad spectrum of architectures, from MLPs and CNNs to Transformers. Our evaluation across eight NVIDIA GPUs, multiple vCPU configurations, and both SSD and NVMe storage tiers demonstrates that host-side factors play a substantial role in shaping end-to-end training performance—an effect that existing GPU-focused predictors systematically overlook. Through extensive cross-fold and leave-one-GPU-out experiments, we show that ORION achieves high predictive accuracy and robust generalization to unseen compute environments. Across all architectures, ORION reduces prediction error by an average of **44.36%**, with improvements ranging from **10.46%** to **92.87%** compared to the state-of-the-art baseline. These results underscore the necessity of incorporating training-host characteristics into modern training-time prediction and highlight the value of unified compute– and data-pipeline modelling for reliable, hardware-aware performance forecasting.

Future Work. While ORION already performs well across heterogeneous training settings, several extensions remain open. First, scaling the framework to large, high-throughput datasets (e.g., ImageNet-21K [7], LAION-5B [36]) would allow evaluation under far heavier storage and decoding pressure, enabling ORION to inform data-pipeline optimization. Second, incorporating distributed training effects such as multi-GPU data parallelism, cross-node communication, and heterogeneous accelerator setups would broaden its applicability to modern cluster-scale workloads. Finally, integrating ORION into automated resource planning tools may support principled cost-performance estimation and help practitioners select balanced GPU, CPU, and storage configurations before training. Overall, ORION provides a foundation for system-aware performance prediction and advances the modelling of end-to-end deep learning training time.

References

[1] Daiyaan Arfeen, Zhen Zhang, Xinwei Fu, Gregory Ganger, and Yida Wang. 2025. Pipefill: Using gpus during bubbles in pipeline-parallel llm training. *Proceedings of Machine Learning and Systems* 7 (2025).

[2] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, et al. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation*. 12–58.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[5] Adam Coates, Andrew Ng, and Honglak Lee. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 215–223.

[6] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2017. Dawnbench: An end-to-end deep learning benchmark and competition. *Training* 100, 101 (2017), 102.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 248–255.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT* (2019).

[9] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2023. Runtime performance prediction for deep learning models with graph neural network. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 368–380.

[11] X Yu Geoffrey, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. 2021. Habitat: A {Runtime-Based} computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 503–521.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. doi:10.1109/CVPR.2016.90

[13] Jim Holtman and Neil J Gunther. 2008. Getting in the zone for successful scalability. *arXiv preprint arXiv:0809.2541* (2008).

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[15] Mohamed A Ibrahim, Shaizeen Aga, Ada Li, Suchita Pati, and Mahzabeen Islam. 2024. JIT-Q: Just-in-time Quantization with Processing-In-Memory for Efficient ML Training. *Proceedings of Machine Learning and Systems* 6 (2024), 46–59.

[16] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[17] Wonkyung Jung, Daejin Jung, Byeongho Kim, Sunjung Lee, Wonjong Rhee, and Jung Ho Ahn. 2019. Restructuring batch normalization to accelerate CNN training. *Proceedings of machine learning and systems* 1 (2019), 14–26.

[18] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. 2018. Predicting the computational cost of deep learning models. In *2018 IEEE international conference on big data (Big Data)*. IEEE, 3873–3882.

[19] Aarati Kakaraparthy, Abhay Venkatesh, Amar Phanishayee, and Shivaram Venkataraman. 2019. The case for unifying data loading in machine learning clusters. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. 1097–1105.

[22] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. 2022. AS-MLP: An axial shifted mlp architecture for vision. In *ICLR 2022-10th International Conference on Learning Representations*.

[23] Mingyu Liang, Hiwot T Kassa, Wenyin Fu, Brian Coutinho, Louis Feng, and Christina Delimitrou. 2025. Lumos: Efficient Performance Modeling and Estimation for Large-scale LLM Training. *Proceedings of Machine Learning and Systems* 7 (2025).

[24] Xiaolong Ma, Feng Yan, Lei Yang, Ian Foster, Michael E Papka, Zhengchun Liu, and Rajkumar Kettimuthu. 2024. MalleTrain: Deep Neural Networks Training on Unfillable Supercomputer Nodes. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*. 190–200.

[25] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2020. Mlperf training benchmark. *Proceedings of Machine Learning and Systems* 2 (2020), 336–349.

[26] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. Analyzing and mitigating data stalls in DNN training. *Proceedings of the VLDB Endowment* 14, 5 (2021), 771–784.

[27] Ramesh Nallapati, Bowen Zhou, Cicero Dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*. 280–290.

[28] Keiron O'shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

[29] Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jaesik Choi, Sam H Noh, and Young-ri Choi. 2020. HetPipe: Enabling large DNN training on (whimpy) heterogeneous GPU clusters through integration of pipelined model parallelism and data parallelism. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 307–321.

[30] Ziqian Pei, Chensheng Li, Xiaowei Qin, Xiaohui Chen, and Guo Wei. 2019. Iteration time prediction for cnn in multi-gpu platform: modeling and analysis. *IEEE Access* 7 (2019), 64788–64797.

[31] Benjamin JJ Pfister, Dominik Scheinert, Morgan K Geldenhuys, and Odej Kao. 2024. Daedalus: Self-Adaptive Horizontal Autoscaling for Resource Efficiency of Distributed Stream Processing Systems. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*. 130–141.

[32] Christian Pinto, Yiannis Gkoufas, Andrea Reale, Seetharami Seelam, and Steven Eliuk. 2018. Hoard: A Distributed Data Caching System to Accelerate Deep Learning Training on the Cloud. *CoRR* (2018).

[33] Alireza Pourali, Arian Boukani, and Hamzeh Khazaei. 2025. PreNeT: Leveraging Computational Features to Predict Deep Neural Network Training Time. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering*. 81–91.

[34] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS Workshop*.

[35] Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. 2018. On Challenges in Machine Learning Model Management. *Data Engineering* (2018), 5.

[36] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. 2022. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems* 35 (2022), 25278–25294.

[37] K Simonyan and A Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society.

[38] Ravi Kumar Singh, Likhith Bandamudi, Shruti Kunde, Mayank Mishra, and Rekha Singhal. 2024. Leftovers for LLaMA. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*. 201–210.

[39] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for

Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Seattle, Washington, USA, 1631–1642. https://nlp.stanford.edu/sentiment/

[40] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[41] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems* 34 (2021), 24261–24272.

[42] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, et al. 2022. Resmlp: Feedforward networks for image classification with data-efficient training. *IEEE transactions on pattern analysis and machine intelligence* 45, 4 (2022), 5314–5321.

[43] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*. PMLR, 10347–10357.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[45] Chuan-Chi Wang, Ying-Chiao Liao, Ming-Chang Kao, Wen-Yew Liang, and Shih-Hao Hung. 2021. Toward accurate platform-aware performance modeling for deep neural networks. *ACM SIGAPP Applied Computing Review* 21, 1 (2021), 50–61.

[46] Jason Wang, Luis Perez, et al. 2017. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit* 11, 2017 (2017), 1–8.

[47] Zhuang Wang, Xinyu Crystal Wu, Zhaozhuo Xu, and TS Eugene Ng. 2023. Cupcake: a compression optimizer for scalable communication-efficient distributed training. In *Proceedings of the Sixth Conference on Machine Learning and Systems (MLSys' 23)*.

[48] Gyeongsik Yang, Changyong Shin, Jeunghwan Lee, Yeonho Yoo, and Chuck Yoo. 2022. Prediction of the resource consumption of distributed deep learning systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 2 (2022), 1–25.

[49] Luca Zancato, Alessandro Achille, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. 2020. Predicting training time without training. *Advances in Neural Information Processing Systems* 33 (2020), 6136–6146.