DeepSim: SOL-Based Explainable GPU Performance Projection with Multi-Task Learning

Mengyue Huang[1], Zhanhui Li[1]*, Tim Purcell[2]*, Mingqiao Li[1], Zhengru Wang[1], Nirmal Saxena[2]
[1] *NVIDIA,* Shanghai, China
[2] *NVIDIA,* Santa Clara, California, USA
Email: ximy@alumni.sysu.edu.cn, {davili, TPurcell, mingqiaol, zhengruw, nsaxena}@nvidia.com

***Abstract*—Accurate kernel-level GPU performance projection is essential for early-stage architecture design when target hardware is unavailable and only high-level specifications are known. Existing methods face a trade-off: simulators reveal bottlenecks but are slow and require unavailable hardware details, while analytical and end-to-end learning models are efficient yet rely on idealized assumptions and provide limited explainability.**

We present DeepSim, an explainable GPU performance projection framework for single-GPU kernels that learns unit-level speed-of-light (SOL) ratios to calibrate bottleneck pressure using multi-task learning. Given target chip specifications and kernel traffic from existing GPUs, DeepSim constructs idealized limiters, predicts per-unit SOL ratios, and maps them to practical limiters reflecting target-chip behavior. Using this representation, DeepSim infers dominant bottlenecks, classifies workloads, and estimates execution time. Across 290K samples and 284 kernels spanning NVIDIA Ampere, Ada, Hopper, and Blackwell GPUs, DeepSim achieves 12.29% mean absolute percentage error on held-out B200 and 14.78% on unseen kernels.

***Index Terms*—GPU, Performance Modeling and Simulation, Multi-Task Learning, Deep Learning**

I. Introduction

Graphics processing units (GPUs) have become a central platform for high-performance computing, artificial intelligence, and data-intensive scientific applications. Their high throughput and massive parallelism make them indispensable in modern supercomputing systems. For example, NVIDIA and Oracle announced a collaboration with the U.S. Department of Energy (DOE) to build the DOE's largest AI supercomputer for scientific discovery, powered by up to 100,000 NVIDIA Blackwell GPUs [1]. More broadly, GPU-accelerated systems now account for a substantial portion of the TOP500 supercomputer list [2].

As computational demand continues to grow, GPU architects must evaluate many candidate designs before hardware is available. Accurate performance projection at this stage is inherently difficult. During early design, there is no access to silicon, complete software stacks, or finalized microarchitectural details such as instruction latencies and scheduling behavior. In many cases, only high-level specifications are available, including execution resources, cache capacities, bandwidths, and clock frequencies. Yet kernel performance emerges from complex interactions among compute pipelines, memory hierarchy components, execution imbalance, and frequency scaling. Practical early-stage projection therefore requires a method that is both scalable and sufficiently accurate across diverse kernels and target-chip configurations.

Existing GPU performance modeling techniques can be broadly divided into analytical modeling, trace-based simulation, hardware emulation, and data-driven modeling. When used for early-stage projection of unavailable target GPUs, each class has important limitations. Analytical models [3]–[16] are efficient, but they typically rely on simplified assumptions and have difficulty capturing fine-grained interactions across pipelines and memory subsystems. Simulators [17]–[32] provide higher fidelity, but they require binaries or detailed hardware specifications that are typically unavailable for future designs, and their runtime cost limits their use in large-scale design space exploration. Hardware emulation is similarly expensive and often impractical at this stage. Data-driven methods [33]–[37] improve efficiency, but existing approaches are typically developed and evaluated on a limited set of existing architectures and workloads, leaving their ability to project performance on unavailable future GPUs insufficiently demonstrated. Moreover, they often provide only end-to-end predictions without exposing the architectural bottlenecks that drive those predictions. As a result, existing solutions still face a fundamental trade-off among fidelity, scalability, and interpretability for early-stage GPU performance projection.

In this work, we propose `DeepSim`, an explainable data-driven framework for kernel-level GPU performance projection on unavailable target GPUs. `DeepSim` currently targets single-GPU applications by modeling the execution time of kernels that run entirely on the GPU. Given target-chip specifications and kernel traffic observed on existing GPUs, `DeepSim` constructs fine-grained throughput-based representations that capture key performance limiters, execution imbalance, and frequency effects with low overhead. These representations are used in a multi-task learning framework to predict unit-level *speed-of-light (SOL) ratios*, which quantify how closely each architectural unit approaches its idealized throughput limit [38]. The predicted *SOL ratio* values bridge idealized limiters and practical bottleneck pressure, enabling `DeepSim` to estimate execution time while also identifying dominant bottlenecks.

A key aspect of `DeepSim` is that it provides result-level interpretability rather than only an end-to-end runtime estimate. By predicting unit-level *SOL ratios* and mapping them to

*Corresponding authors

SC26, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-4789-7/26/$31.00 ©2026 IEEE

practical limiters, DeepSim attributes performance behavior to specific parts of the GPU execution and memory hierarchy. This representation helps explain why a workload is compute-bound, cache-bound, bandwidth-bound, or limited by other effects, and makes the projected performance easier to analyze in both workload studies and architectural exploration.

To evaluate robustness and generality, we develop and study DeepSim using a diverse dataset spanning NVIDIA compute accelerators (GA100, H100, GH200, B200), data-center GPUs (L40), and consumer GPUs (GeForce RTX 5080, GeForce RTX 5090), covering 290,000 samples from 284 kernels across multiple application domains. We evaluate DeepSim on held-out chips and unseen kernels, with the primary analysis focused on a held-out B200 setting that emulates projection to an unavailable target GPU. The results show that DeepSim achieves strong predictive accuracy while retaining useful bottleneck-level interpretability.

The main contributions of this work are summarized as follows:

- We propose DeepSim, an explainable GPU performance projection framework that predicts unit-level *SOL ratios* using multi-task learning and uses them to estimate kernel execution time on unavailable target GPUs.
- We introduce a fine-grained feature modeling framework that captures pipeline bottlenecks, execution imbalance, and frequency scaling through throughput-oriented representations derived from chip specifications and kernel traffic.
- We build DeepSim using a dataset of 290,000 samples spanning multiple NVIDIA GPU generations, and evaluate it on 284 representative kernels. DeepSim achieves strong accuracy on held-out target chips and unseen kernels, with the evaluation primarily centered on a held-out B200 setting.
- We show that DeepSim provides useful bottleneck-level insights for program analysis and supports chip design space exploration for next-generation GPU development.

II. Background and Motivation

A. Related Work

GPU performance modeling is central to performance analysis, bottleneck diagnosis, and early-stage architecture projection. Prior work falls into three broad classes: analytical models, simulation-based methods, and learning-based methods, each with different trade-offs in fidelity, efficiency, generality, and interpretability.

Analytical models [3]–[16] provide lightweight, efficient estimates by modeling key throughput or bandwidth constraints. Representative examples include warp-level latency-hiding analysis [5] and roofline-style projection for upcoming GPUs [4]. Their speed makes them attractive for large-scale design exploration, and their structure is often physically interpretable. However, they typically rely on hand-designed abstractions of software–hardware interactions and simplified assumptions about execution. As GPU architectures become more complex, these abstractions often fail to capture fine-grained pipeline effects, execution imbalance, and cross-unit interactions, limiting accuracy and generality. Some analytical methods [4], [5], [14]–[16] also require stronger target-side information, such as target-specific compiled kernels or assembly-level analysis, which may be difficult to obtain in strict early-stage design.

Simulation-based approaches [17]–[32] achieve higher fidelity by modeling execution in greater architectural detail, making them useful for bottleneck analysis and detailed design studies. Representative simulators such as NVArchSim [21] and AccelSim [22] capture complex architectural behavior across large workloads. However, detailed simulators typically require executable binaries, traces, or low-level microarchitectural parameters that may be unavailable for future GPUs. Their runtime overhead also limits rapid design space exploration. Supporting a new architecture usually requires substantial implementation, calibration, and validation effort, and public support for the latest GPU generations may lag hardware release. Hybrid analytical–simulation approaches [24], [39]–[41] reduce some of this cost, but they still depend on trace collection or interval-based execution analysis and therefore remain less scalable than lightweight projection methods.

Learning-based approaches provide an alternative by learning performance characteristics directly from data. Traditional machine learning methods, including linear regression [42]–[44], support vector regression [45], tree-based models [36], [46]–[48], multi-layer perceptrons [34], and hybrid and model-selection approaches [33], [35], [37], [49], [50], can be efficient at inference time and often outperform hand-crafted analytical formulas when sufficient training data are available. More recent deep learning approaches, including neural networks [51], LSTM-based models [52], representation learning [53], and LLM-based predictors [54], can capture increasingly complex nonlinear relationships in GPU performance. These methods improve inference efficiency and can capture nonlinear performance behavior, but most are trained and evaluated on limited sets of existing architectures or workload domains. Their ability to project performance on unavailable target GPUs is therefore insufficiently demonstrated. In addition, many learning-based models predict end-to-end runtime or performance metrics directly, with limited visibility into the architectural bottlenecks that drive the prediction.

Adjacent CPU studies have likewise shown the effectiveness of deep learning for performance modeling, including basic-block throughput prediction and joint program–architecture representation learning [55], [56]. However, these works address settings that differ substantially from early-stage GPU projection, typically assuming existing target processors and lower-level program representations or prediction targets.

Prior work therefore does not fully satisfy the combination of requirements for early-stage GPU projection: scalability across large design spaces, robustness to unseen target chips and kernels, weak dependence on target-side software artifacts, and bottleneck-level interpretability. These gaps motivate a model that preserves the efficiency of data-driven predic-

tion while operating under weak target-side assumptions and grounding its outputs in architecture-aware, unit-level performance representations.

B. GPU: A Throughput-Oriented Architecture

Figure 1 shows a simplified NVIDIA GPU architecture as a representative abstraction. We use NVIDIA terminology for clarity, but the method is not NVIDIA-specific: real GPU designs are more complex, yet they share broadly similar structures across vendors and generations. A modern GPU typically comprises tens to hundreds of Streaming Multiprocessors (SMs) and a memory hierarchy that includes L1 caches (both the hardware-tagged L1 data cache and the software-managed shared memory on NVIDIA GPUs), a crossbar interconnect (XBAR), L2 caches, and DRAM. Each SM can execute hundreds of single- or double-precision operations per cycle through multiple execution pipelines. One or more SMs form a Texture Processing Cluster (TPC), and several TPCs form a Graphics Processing Cluster (GPC). When a GPU kernel is launched, threads are grouped into warps—the minimal unit of execution. Each warp is scheduled on an SM and executes in a SIMT (Single Instruction, Multiple Threads) manner. Tens of warps can be active concurrently on an SM. When a warp stalls due to data dependencies or memory operations, it is replaced by an eligible warp that is ready to execute. This warp-switching mechanism, combined with the extensive parallelism of tens of active warps, allows GPUs to sustain high throughput.

C. Performance Limiters

Achieving peak theoretical compute or memory throughput with a real program is difficult because execution passes through many stages, each with its own constraint. Examples include instruction issue throughput and register file bandwidth within the SM, request and data-transfer throughput in the XBAR, and bandwidth limits in L1 and L2, among others. We refer to these throughput-constraining pipeline components as ***performance limiters***. An *SOL ratio* approaching 100% indicates that the corresponding limiter is nearly saturated and is therefore a likely bottleneck, whereas a low *SOL ratio* indicates underutilization.

Modern GPUs expose many performance limiters through NVIDIA Nsight Compute [57]. Different workloads stress different subsets of these limiters, which makes performance analysis difficult. From 284 kernels across diverse applications, we identify 20 frequently occurring limiter types, detailed in Section III-B. `DeepSim` expands them into limiter-derived input features, and the framework can be extended to additional GPU limiters.

Our study shows that prominent limiters arise mainly from SM, L1, L2, XBAR, and DRAM, as illustrated in Figure 1. In `DeepSim`, we explicitly model four throughput-oriented units: SM, L1, L2, and DRAM. We absorb XBAR transfer effects into the L1/L2-related limiters rather than treating XBAR as a separate modeled unit, because its behavior is tightly coupled to data movement between L1 and L2. We therefore classify workloads into four throughput-bound categories and an *Others* category. The *Others* category corresponds to kernels with uniformly low *SOL ratios* across all throughput-oriented units, suggesting latency-bound behavior rather than a dominant throughput bottleneck.

- **SM-bound**: Workloads constrained by limiters within the SM, such as instruction issue throughput, computation pipeline throughput, and register file bandwidth.
- **L1-bound**: Workloads constrained by limiters within the L1, such as cache or shared memory bandwidth.
- **L2-bound**: Workloads constrained by limiters within the L2, such as cache bandwidth.
- **DRAM-bound**: Workloads constrained by limiters within the DRAM, such as global memory bandwidth.
- **Others**: Workloads for which no throughput-oriented unit is clearly dominant, typically indicating latency-bound behavior caused by insufficient parallelism or data dependencies.

D. Performance Limiter Dynamics

To illustrate cross-chip variation in bottleneck behavior, Figure 2 shows the distribution of dominant limiters for a set of kernels across seven chips. The dominant limiter changes substantially across architectures. For example, *DRAM: reads & writes* accounts for 54.36% on L40 but 34.68% on H100, while *L1:wavefronts* reaches 20.97% on GH200 but only 2.74% on GeForce RTX 5080. For the same kernel, the dominant limiter also often differs across chips: 3.4% of kernels have different dominant limiters on five or more chips, 7.7% on four, 16.4% on three, and 36.2% on two. This variability becomes even stronger when secondary limiters are considered, suggesting that fixed analytical abstractions may struggle to capture such irregular cross-architecture behavior, whereas learning-based methods are better suited to model these nonlinear relationships.

III. DeepSim Design

A. Overview

Figure 3 outlines the `DeepSim` workflow. During inference, the goal is to predict the execution time of Kernel A on Chip T, where Chip T may be a forthcoming or otherwise inaccessible architecture. `DeepSim` first extracts limiter-aware features from two inputs: (1) the architectural specifications of Chip T and (2) the traffic characteristics of Kernel A profiled on an available base chip, Chip X. These inputs characterize the ideal throughput pressure induced by the kernel on the target architecture, but they do not directly capture how much of that idealized pressure can be realized in practice. Thus, base-chip traffic is used only as an initial proxy for target-side demand; `DeepSim` relies on target-side *SOL ratio* prediction to calibrate this proxy into practical bottleneck pressure. `DeepSim` therefore predicts unit-level *SOL ratios* for SM, L1, L2, and DRAM using a multi-task learning model. The predicted *SOL ratios* quantify the gap between ideal and realized performance for each unit, and thus provide an

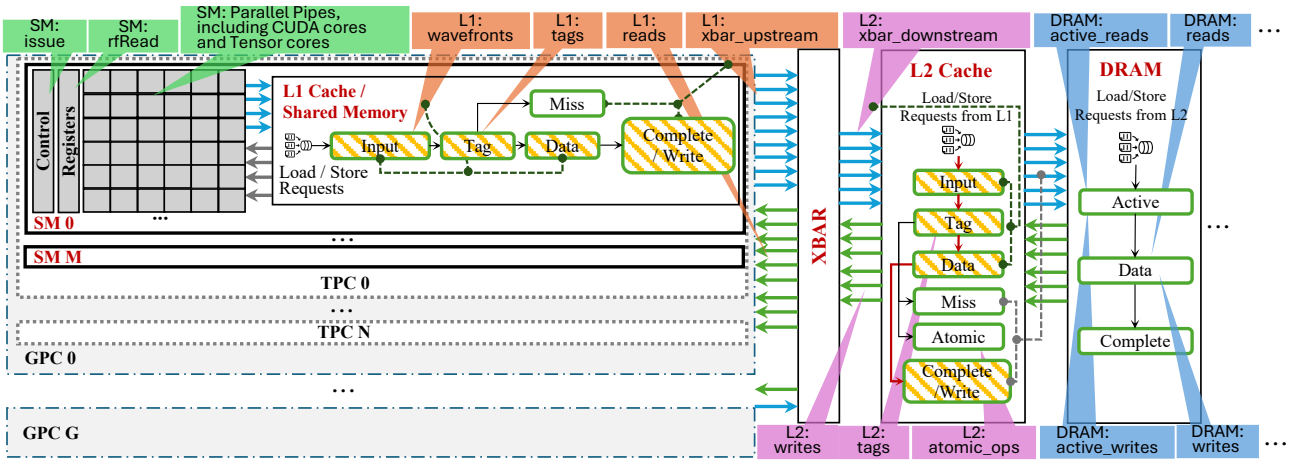


Fig. 1: GPU architectures include four throughput-oriented units explicitly modeled in this work: SM, L1 (and shared memory), L2, and DRAM, together with the XBAR interconnect between L1 and L2. The execution stages within these units, together with the detailed performance limiters shown in the external callout annotations, are described in Section III-B.

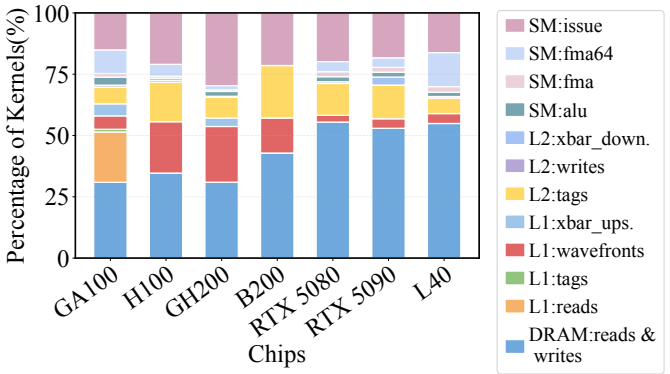


Fig. 2: Distribution of per-kernel dominant limiters across seven chips for a set of kernels. The percentage denotes the fraction of kernels for which a given limiter appears as the top limiter (Section III-B).

architecture-aware intermediate representation between traffic-derived limiters and actual execution behavior on Chip T. The predicted *SOL ratios* are then used for workload classification and practical-limiter construction. Based on the dominant predicted unit, `DeepSim` assigns the workload to one of five categories: SM-bound, L1-bound, L2-bound, DRAM-bound, or *others*. A dedicated dual model is then applied to predict execution time.

During training, multiple chips and kernels are profiled to collect performance reports. In each report, Chip T and Chip X may refer to the same device. A given Chip T can also be paired with different base chips as Chip X, which provides data augmentation. This design exposes the model to the same kernel under multiple chip pairings and the resulting variation in traffic-derived features and realized performance. As a result, the model is trained under a broader range of cross-chip projection conditions and is better positioned to handle base-target mismatch during inference.

B. FTIM (Fine-Grained Traffic-Based and Imbalance-Aware Modeling)

FTIM characterizes four GPU units, SM, L1, L2, and DRAM, using fine-grained, imbalance-aware, and frequency-sensitive limiters (Figure 1).

1) Performance Limiters: For each unit, the ***ideal limiter*** is the observed traffic normalized by the peak throughput of a pipeline stage, and the ***practical limiter*** scales the ideal limiter by the corresponding *SOL ratio*. Here, *traffic* denotes instruction count for compute units and transferred bytes for memory units.

For SM, we model the *issue* stage, CUDA-core pipelines for *alu*, *fma*, *fma64*, *xu*, and *mio*, the tensor-core *dmma* pipeline, and *register read*, yielding eight limiters. For the HPC-centric workloads considered in this work, modeling *dmma* is sufficient in most cases; the model can be extended to additional tensor-core pipelines when needed (Section V-D). For the memory hierarchy, both L1 and L2 include *input*, *tag*, *data*, *miss handling*, and *write*; L2 additionally includes *atomic*, and DRAM uses a single *active* stage for read and write operations. The L1 *input* stage accounts for bank conflicts and wavefront formation, while XBAR serves as the interconnect. These memory-hierarchy units contribute twelve limiters.

2) Imbalance-Aware and Frequency-Sensitive: Cooperative Thread Array (CTA) placement and per-CTA traffic are often uneven, which creates heterogeneous pressure across unit instances. To capture this effect, we derive two features for each limiter: an ***average limiter*** from mean traffic across instances and a ***maximum limiter*** from the most heavily loaded instance. Frequency sensitivity is incorporated by converting cycle-based throughput to time using the corresponding GPC or DRAM frequency, so limiter values scale with operating frequency.

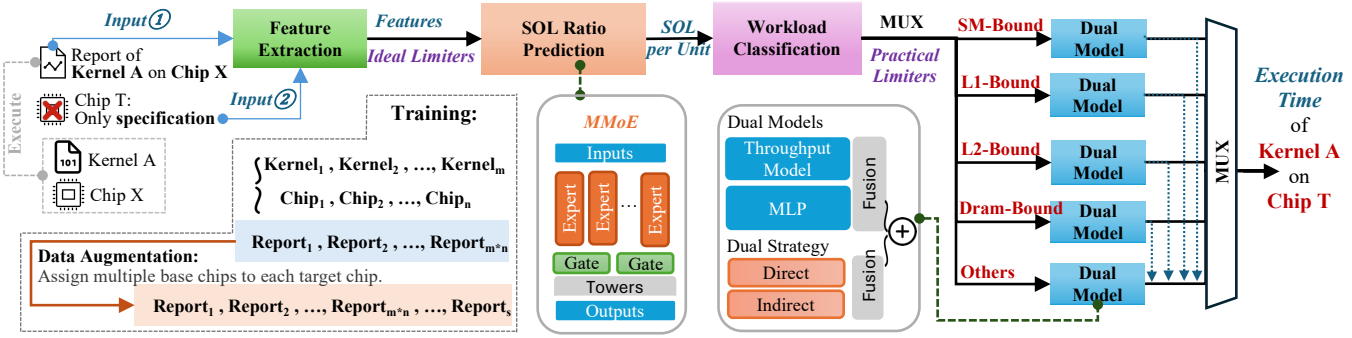


Fig. 3: Workflow of DeepSim. During inference, DeepSim takes the architectural specifications of Chip T and the performance reports of Kernel A as inputs, followed by FTIM feature extraction, SOL prediction, workload classification, and dual-model execution time prediction. During training, DeepSim collects profiling reports from multiple chips and applies data augmentation to enrich the training dataset.

3) Extensible Limiters and Architectural Units: To accommodate evolving GPU architectures, FTIM uses an extensible abstraction for both limiters and architectural units. New limiters can be added by identifying the corresponding traffic metric and peak stage throughput. New units can be incorporated by defining their pipelines and extracting the associated traffic features, which are then encoded with the same average and maximum formulations. This modular design allows FTIM to adapt to new GPU components without changing the core framework. Section V-D gives one such extension.

4) Feature Extraction: The input to FTIM consists of kernel traffic and chip configuration. Chip configuration is available from architectural specifications and does not require access to the target hardware. Kernel traffic is collected on an available base chip, Chip X, by profiling Kernel A and using the resulting traffic profile as a proxy for its behavior on target Chip T. Based on these inputs, DeepSim extracts 46 features: 20 average limiters, 20 maximum limiters, and six architectural descriptors: L1 cache size (and associated shared memory), L2 cache size, global memory capacity, number of launched CTAs, number of launched warps, and memory hierarchy type.

C. Multi-Task Learning for SOL Ratio Prediction

SOL ratio prediction forms the core of the overall performance modeling framework. The *SOL ratio* predictor takes as input all features extracted in Section III-B4, using the *ideal* versions of both the average and maximum limiters. It outputs the *SOL ratios* for SM, L1, L2, and DRAM, which are subsequently used for workload classification and practical limiter computation, capturing the actual utilization of Kernel A on Chip T.

Each unit's *SOL ratio* is governed by a heterogeneous set of limiters with distinct operational characteristics. For example, the SM *SOL ratio* depends on issue throughput, register file bandwidth, and the utilization of multiple execution pipelines, whereas memory units (e.g., L1 and L2) are more sensitive to access locality, bank conflicts, and cache miss behavior. These factors differ in magnitude, noise characteristics, and sensitivity, resulting in inherently task-specific response functions. A fully shared learning model may therefore cause negative transfer across tasks, degrading predictive accuracy. Meanwhile, units exhibit strong interactions. Their effective throughput and stall behavior are tightly coupled through the memory hierarchy and pipeline backpressure. For instance, the L2 *SOL ratio* depends on incoming traffic from L1 and responses from DRAM, while SM utilization may be constrained by memory subsystem delays. Such cross-unit dependencies create complex, workload-dependent correlations that independent models fail to capture.

To address these challenges, we adopt a Multi-Task Learning (MTL) framework based on the Multi-gate Mixture-of-Experts (MMoE) architecture [58]. MMoE allows tasks to dynamically combine shared experts through task-specific gating networks, enabling the model to learn both shared performance patterns and unit-specific behaviors. In particular, the shared experts capture common performance dynamics across units, while the gating mechanism allows each task to selectively leverage them to model unit interactions. This design mitigates negative transfer, improves generalization, and enables the model to identify cross-unit performance interactions. Alternative model architectures are evaluated for comparison, as described in Section V-F2.

D. Dual Model for Elapsed Cycle Prediction

Using the practical limiters derived from the estimated *SOL ratios* and ideal limiters, together with additional architectural features, DeepSim constructs a prediction model tailored to each workload category. The estimated *SOL ratios* calibrate the traffic-derived ideal limiters into practical limiters, so the resulting features better reflect realized pressure on Chip T rather than idealized throughput demand alone. This correction is intended to reduce the mismatch between base-chip traffic observations and target-chip execution behavior, especially when cross-chip differences shift the dominant bottleneck.

The proposed dual model uses a two-level integration strategy. At the first level, it combines a Multi-Layer Perceptron (MLP) with a throughput model: the MLP captures nonlinear correlations in kernel behavior, while the throughput model

provides hardware-aware structure and helps preserve physical interpretability. At the second level, dual model combines two complementary approaches: a *direct* approach, which predicts the elapsed cycles of Chip T directly from the input features, and an *indirect* approach, which estimates the performance scaling ratio between Chip X and Chip T and then derives the predicted execution time for Chip T.

The output of the *direct* approach is formulated as a linear combination of the MLP and throughput model outputs. The final prediction is obtained as a weighted combination of the *direct* and *indirect* results. These weights are learned during training by minimizing prediction error, allowing dual model to adaptively balance the contributions of the MLP, throughput, *direct*, and *indirect* components. Dedicated dual model instances are constructed for the SM, L1, L2, and DRAM categories. For the *others* category, in addition to its own dual model, `DeepSim` also aggregates predictions from the other units to produce the final execution-time estimate. This hierarchical fusion allows the model to use both corrected unit-level signals and cross-chip performance trends, which can improve prediction accuracy and generalization across diverse workloads and architectures.

E. SOL ratio-based Workload Classification

Workloads are classified into five categories based on the predicted *SOL ratio*. A unit exhibiting a high maximum *SOL ratio* indicates that the workload is throughput-bound on that unit, whereas consistently low values suggest a latency-bound behavior. If no unit emerges as dominant (i.e., all units have uniformly low *SOL ratios*), the workload is assigned to the *others* category, which typically corresponds to latency-bound kernels. Otherwise, the unit with the highest *SOL ratio* determines the throughput-bound category. It is important to note that the same kernel may belong to different categories on different chips due to variations in chip configuration.

IV. Implementation

A. Workloads

`DeepSim` uses a comprehensive benchmark suite summarized in the first two columns of Table I, covering 11 domains and 284 kernels, including molecular dynamics, computational fluid dynamics, sparse linear algebra, genomics, quantum computing, and electronic design automation. The suite combines public applications and benchmarks, including AMGX [59], GROMACS [60], LAMMPS [61], SPECFEM3D [62], QUDA [63], TPC-H [64], and CP2K [65], with additional confidential application sources. Measured GPU-kernel runtimes in this corpus range from approximately 40K to 329M cycles, covering both short and long-running kernels. In addition to full-application benchmarks, `DeepSim` employs synthetic microbenchmarks to isolate hardware limits, focusing on instruction throughput and memory bandwidth. Instruction tests saturate individual or mixed SM pipelines (e.g., FMA and FMA64), while bandwidth tests stress load/store throughput across L1, L2, and DRAM. These microbenchmarks characterize peak subsystem capabilities and complement application benchmarks.

B. Chips

`DeepSim` covers seven representative GPUs: GA100 [66], L40 [67], H100 [68], GH200 [69], GeForce RTX 5080 [70], GeForce RTX 5090 [70], and B200 [71]. Although B200, GeForce RTX 5080, and GeForce RTX 5090 all belong to NVIDIA's Blackwell family, they represent distinct performance-modeling targets. B200 is a compute accelerator with distinct CUDA compute capability and SASS target, HBM3e memory subsystem, higher memory bandwidth, and larger per-SM L1/shared-memory capacity, whereas GeForce RTX 5080 and GeForce RTX 5090 are RTX-class GPUs with GDDR7 memory and varying SM configurations and memory-subsystem characteristics. To capture architectural and runtime variability, both GPC and DRAM frequencies are varied for each chip, yielding 33 base chip configurations. Multi-Instance GPU (MIG) [72] is used on H100 to create additional partitions, such as half-instance and single-GPC modes, each of which is likewise profiled across frequency points.

C. Training Schemes

Hardware performance counters are collected using NVIDIA Nsight Compute (NCU) [73]. Profiling our full application corpus across the collected chip configurations yields over 80K raw samples. We split the corpus *at the application level*: every kernel originating from a held-out application, including all of its versions, is excluded from training, which is stricter than a per-kernel split. The split leaves 80% of kernels for training and 20% held out for testing. Data augmentation is then applied only to the training split. Exchanging kernel traffic characteristics across chips expands the training set to 160K samples, and additional synthetic kernels derived from IPC and memory tests further enlarge it to 290K samples. In addition, GA100 and B200 are held out entirely from training, and no profiling reports from these chips are used to construct training samples. This setup enables evaluation on held-out target chips and held-out kernels (Section V-A).

V. Evaluation

We evaluate `DeepSim` primarily on the held-out target chip B200, focusing on accuracy, generality, explainability, extensibility, and comparison with representative methods. We also present component analyses and additional results on other chips, including GA100.

A. Accuracy and Generality

We evaluate `DeepSim` from three perspectives: overall accuracy on the held-out target chip, held-out kernels, and different workload categories. The evaluation metric is mean absolute percentage error (MAPE), which quantifies the average relative deviation between predicted and silicon-measured execution times (SI).

TABLE I: Benchmark domains with corresponding kernel counts and their distribution across five categories (SM, L1, L2, DRAM, and others) on B200. The MAPE for each domain is reported in the last column, and the category-wise MAPE is summarized in the last row.

Domain	Kernel Number	SM-bound	L1-bound	L2-bound	DRAM-bound	Others	MAPE
Molecular Dynamics & Chemistry	75	32	29	8	6	0	8.63%
Computational Fluid Dynamics	33	4	21	0	8	0	13.59%
Database & Data Analytics	33	13	14	2	4	0	12.99%
Sparse Linear Algebra	32	6	5	2	18	1	15.32%
Genomics & Bioinformatics	7	4	1	0	2	0	10.04%
Geophysics & Seismic Modeling	24	2	13	2	6	1	13.09%
Plasma & Fusion Simulation	21	0	12	2	4	3	16.27%
Finance & Risk Modeling	16	9	5	0	0	2	16.10%
Computer-Aided Engineering	7	4	0	0	2	1	12.80%
Quantum Computing	12	5	1	1	5	0	11.77%
Synthetic Microbenchmarks	24	5	9	2	8	0	10.91%
Total Number	284	84	110	19	63	8	**12.29%**
MAPE	**12.29%**	9.97%	9.59%	17.39%	17.48%	20.80%	

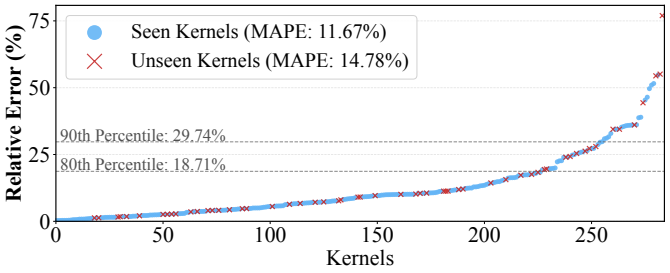


Fig. 4: Relative errors of all kernels on B200, including both training-seen and held-out kernels, together with the corresponding MAPE. The 80th and 90th percentiles are indicated by dashed lines.

1) On the Held-Out Target Chip: We evaluate all kernels on the held-out target chip B200, using H100 as the base chip during inference. Since B200 is excluded from training, this setting emulates performance projection to an unavailable target chip. It includes both kernels seen during training on other chips and kernels held out entirely from training. The resulting overall MAPE of **12.29%** summarizes end-to-end prediction accuracy on B200. Table I summarizes the kernel distribution across five workload categories for each domain on B200, and reports the corresponding MAPE in the last column. The highest domain-level MAPE values—16.27% for *Plasma & Fusion Simulation* and 16.10% for *Finance & Risk Modeling*—coincide with the largest shares of *others* kernels in those domains. Since *others* kernels are harder to model, these domains exhibit lower accuracy. Section V-A3 discusses this category in more detail.

Beyond average error metrics, it is also important to assess the overall distribution of prediction errors. Figure 4 illustrates the error distribution across all kernels on B200, showing that 80% of kernels exhibit errors below 18.71%, and 90% remain within 29.74%. These results indicate that `DeepSim` maintains consistent accuracy across a wide range of kernels.

2) Across Held-Out Kernels: Among the 284 kernels, 57 (20%) are held out before augmentation and are never used to construct training samples. Evaluated on B200 with H100 as the base chip, these held-out kernels yield an average MAPE of 14.78%. The red crosses in Figure 4 mark their errors. Among the held-out kernels, 77% (44 kernels) exhibit errors below 20%, indicating good kernel-level generalization. The maximum error of 77.0% occurs for a kernel in the *others* category, characterized by uniformly low *SOL ratios*, where intrinsic execution-time variability leads to unstable predictions, as further analyzed in Section V-A3. Three kernels exhibit errors between 40% and 60%. In these cases, B200's larger L2 cache leads to higher cache hit rates and correspondingly different L2-side traffic behavior from that observed on the base chip H100. As discussed in Section V-C2, *SOL ratio* modeling helps mitigate cross-chip traffic shift, but memory-hierarchy behavior remains harder to capture when traffic differences are coupled with cache-hit/miss transitions. Although `DeepSim` incorporates cache size information in both *SOL ratio* prediction and dual model, the resulting features capture this effect only partially, leading to higher prediction deviations. This suggests that a more detailed cache model could further improve accuracy in such cases.

3) Across Different Workload Categories: The last row of Table I reports category-wise MAPE values of 9.97%, 9.59%, 17.39%, and 17.48% for SM-, L1-, L2-, and DRAM-bound workloads, respectively. The lower errors for SM- and L1-dominated kernels suggest that `DeepSim` captures throughput-oriented bottleneck pressure well, while the higher errors for L2- and DRAM-bound kernels reflect the greater sensitivity of memory-side behavior to cache-hit and cache-miss variation.

Eight kernels (2.82%) fall into the *others* category, and they yield an average MAPE of 20.8%. Within this small group, three kernels have errors below 3.0% and three remain below 16%; the larger errors are concentrated in two cases, including the maximum error of 77.0% for a kernel launching only 23 CTAs. This pattern is consistent with latency-dominated

kernels being less governed by a single throughput bottleneck and more sensitive to limited parallelism. A more detailed cache model may further improve prediction of their unhidden latencies.

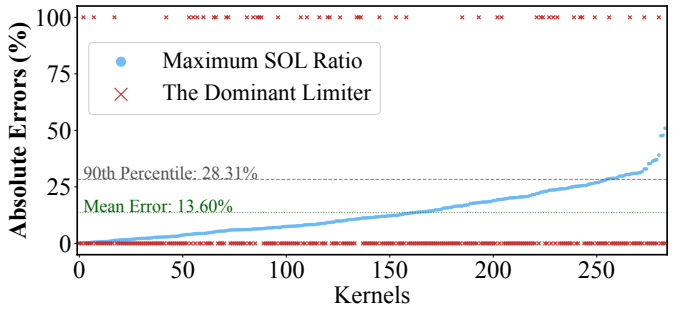


Fig. 5: *SOL ratio* and top-limiter projection error for `DeepSim`. Blue points show the absolute error of the maximum *SOL ratio* across the four units, with mean and 90th-percentile errors per kernel. Red crosses indicate dominant-limiter matching (0: perfect match; 100%: mismatch).

B. Explainability

The predicted *SOL ratio* of each unit and the corresponding top limiter provide a basis for explainability and help users interpret the model's predictions. We therefore evaluate both *SOL ratio* accuracy and top-limiter identification.

1) SOL Ratio Prediction: The blue points in Figure 5 represent the absolute errors of the maximum *SOL ratio* among the four units for each kernel, where the maximum *SOL ratio* corresponds to the dominant bottleneck unit. The mean error is 13.60%, with 90% of kernels showing errors below 28.31%. The mean absolute errors of predicted *SOL ratio* for SM, L1, L2, and DRAM are 12.64%, 12.06%, 11.02%, and 10.91%, respectively. 88.57% of kernels exhibit prediction errors below 30% for all four units, indicating the high accuracy of *SOL ratio* prediction.

2) Dominant Limiter Prediction: The red crosses in Figure 5 indicate whether the predicted dominant limiter matches the silicon result for each kernel, where 0 denotes a perfect match and 100% denotes a mismatch. Overall, `DeepSim` achieves an accuracy of 83.10% in identifying the target dominant limiter. Among the most frequent mismatches, when the true top limiter is *L1:reads*, `DeepSim` predicts *L1:wavefronts* as the dominant limiter in 23 of 30 cases. Further analysis indicates that these two limiters are closely related, as both are associated with L1-bound bottlenecks. Importantly, `DeepSim` reports all modeled limiters to users, which prevents misinterpretation and helps minimize the impact of such mismatches.

C. SOL-Ratio Modeling Analysis

1) Necessity of Target SOL-Ratio Prediction: `DeepSim` predicts target-chip *SOL ratios* as an intermediate representation and applies a practical-limiter correction before runtime estimation. To quantify the contribution of this representation, we construct a variant that bypasses both the SOL-ratio prediction and practical-limiter correction stages during training and inference, instead directly predicting runtime from raw traffic-derived features. Table II reports the MAPE of both schemes, categorized by the dominant limiter of each kernel. The variant consistently exhibits lower accuracy across all categories, increasing the overall MAPE from 12.29% to 28.08%. The degradation is particularly pronounced for L2-bound kernels, where MAPE increases from 17.39% to 79.02%, indicating that their runtime behavior is dominated by queueing and latency-hiding effects that cannot be fully captured by traffic counts alone.

TABLE II: MAPE with and without the *SOL ratio*-based pipeline across five workload categories.

Schemes	SM	L1	L2	DRAM	Others	Overall
with *SOL ratio*	9.97%	9.59%	17.39%	17.48%	20.80%	12.29%
without *SOL ratio*	22.09%	27.37%	79.02%	27.49%	33.87%	28.08%

TABLE III: Ablation of target-side *SOL ratio* prediction and learned runtime modeling on B200.

SOL ratio source	Runtime model	B200 MAPE
Predicted *SOL ratio*	dual model	12.29%
Predicted *SOL ratio*	Analytical-only	20.06%
Base-chip *SOL ratio* as proxy	dual model	22.01%
Base-chip *SOL ratio* as proxy	Analytical-only	30.06%

The pipeline contains two learned components: target-*SOL ratio* prediction and dual model elapsed-cycle modeling. To isolate their contributions, we independently replace each component on B200 (Table III), substituting an analytical runtime model for dual model and measured H100 *SOL ratios*—the closest available base-chip proxy—for the predicted target *SOL ratios*. Each substitution causes substantial accuracy degradation. Replacing dual model increases MAPE from 12.29% to 20.06%, indicating that the learned runtime model captures execution effects beyond the analytical formulation. Replacing predicted *SOL ratios* causes an even larger degradation: MAPE increases to 22.01% and dominant-limiter accuracy drops from 83.10% to 60.52%, showing that bottleneck pressure shifts across chip generations and that base-chip measurements cannot accurately represent target-chip behavior. Removing both components results in 30.06% MAPE, close to the combined degradation of the two individual removals, suggesting that they address largely independent sources of error. Notably, this configuration is *less* accurate than removing the entire *SOL ratio* pipeline (28.08% in Table II), indicating that an inaccurate intermediate representation can introduce larger errors than omitting the representation entirely. This highlights the importance of predicting, rather than directly transferring, *SOL ratio* representations. Together, these results establish the *SOL ratio* representation as more than a modeling convenience: it enables `DeepSim` to transfer bottleneck behavior across chip generations instead of relying solely on

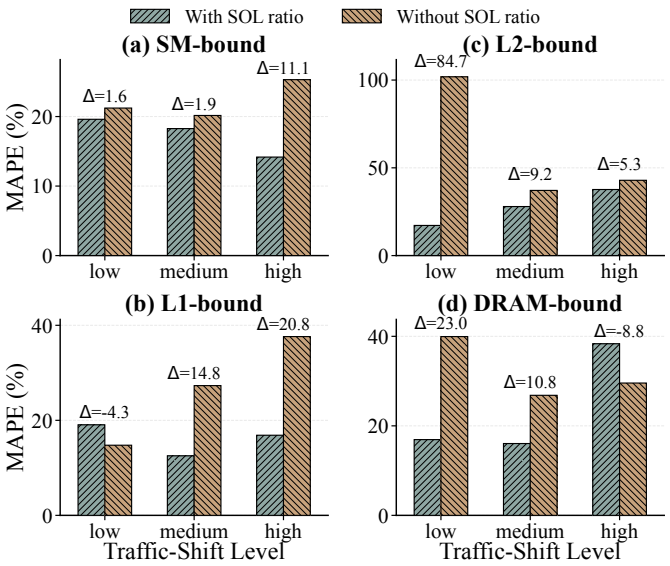


Fig. 6: MAPE of `DeepSim` with and without the *SOL ratio*-based pipeline across traffic-shift levels from H100 to B200. Traffic shift is the normalized H100-to-B200 traffic change for each kernel's dominant unit; kernels within each unit are ranked by this metric and grouped into low, medium, and high levels using the 1/3 and 2/3 quantiles. Positive Δ denotes the error reduction from *SOL ratio*.

traffic-based runtime extrapolation.

2) Robustness of SOL-Ratio Modeling to Traffic Shift: In cross-generation GPU projection, base-chip traffic provides only an initial proxy for target-side demand and may differ substantially from traffic on the target GPU, especially when cache capacity and memory-hierarchy behavior change across chips. To test whether the *SOL ratio*-based pipeline improves robustness to this mismatch, we group kernels by the traffic-shift level of their dominant unit and compare the MAPE of `DeepSim` with and without it in Fig. 6. For compute-side and upper-memory units, the benefit tends to grow with traffic shift. For SM-bound kernels, the relative MAPE reduction increases from 1.6% in the low-shift group to 1.9% in the medium-shift group and 11.1% in the high-shift group. L1-bound kernels show a similar trend: the *SOL ratio*-based pipeline increases MAPE by 4.3% in the low-shift group, but reduces it by 14.8% and 20.8% in the medium- and high-shift groups. This suggests that when traffic mismatch increases but the dominant bottleneck remains similar, unit-level *SOL ratio* provides a more stable intermediate representation than direct runtime prediction alone.

For memory-hierarchy units, the trend is less stable. For L2-bound kernels, the *SOL ratio*-based pipeline reduces MAPE by 84.7%, 9.2%, and 5.3% in the low-, medium-, and high-shift groups. For DRAM-bound kernels, it reduces MAPE by 23.0% and 10.8% in the low- and medium-shift groups, but slightly increases it in the high-shift group (-8.8%). This suggests that large memory-side traffic shifts reflect not only traffic-magnitude changes, but also cache-hit/miss transitions and broader memory-behavior changes. In such cases, unit-level *SOL ratio* captures target-side behavior only partially. Overall, the *SOL ratio*-based pipeline mitigates cross-chip traffic shift, but its benefit is unit-dependent: it is most consistent for SM- and L1-bound kernels and less stable under large memory-side shifts, where more explicit modeling may be needed.

D. Extensibility to DL Kernels

Our model is initially trained without deep learning (DL) kernels. To evaluate extensibility, we use FP16 (HMMA) Tensor Core kernels as a case study, while the design remains applicable to emerging precisions such as FP8, FP6, and FP4. Eight representative HMMA-based GEMM kernels are extracted from open-source large language models, with a focus on DeepSeek [74]. These kernels follow the V3.1 configuration [75] (batch size 32 and sequence length 2048) and are implemented using CUTLASS [76]. Because the model does not see HMMA kernels during initial training, direct application yields a baseline MAPE of 50.5%. We then fine-tune the model with 119 additional samples from L40, H100, and GH200, while holding out three of the eight kernels for testing. After fine-tuning, MAPE drops from 50.5% to 8.6%, as shown in Figure 7. For the seen kernels, all MAPE values remain below 15%; for the held-out kernels, two are below 10%, and the remaining one reaches 24.56%.

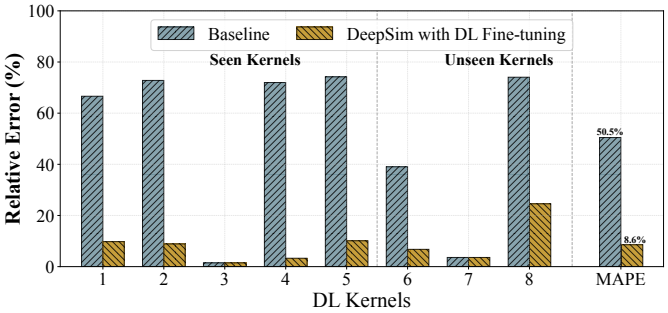


Fig. 7: Comparison of relative errors before and after fine-tuning `DeepSim` with HMMA kernels, including the corresponding MAPE. Eight kernels are included, of which five were seen during fine-tuning and three are unseen.

E. Comparison with Representative Methods

1) Comparing with simulators and analytical models: We compare `DeepSim` with representative GPU performance models from two related paradigms: the simulator NVArchSim [21] and the lightweight analytical model [4]. These methods target related but different settings. `DeepSim` is designed for early-stage projection using only high-level target-chip specifications and traffic from an available base GPU. In contrast, NVArchSim and the analytical model assume stronger target-side availability, such as simulator-specific target modeling support for NVArchSim and target-specific compiled kernels or instruction-level information for the analytical model. They are therefore less applicable when the target GPU

is unavailable and only architectural specifications are known. We use them as reference methods for post-hoc quantitative comparison rather than as methods evaluated under the same input information as DeepSim.

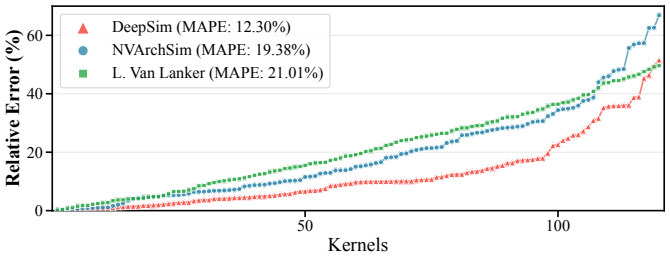


Fig. 8: Comparison of relative errors among NVArchSim [21], the analytical model proposed by L. Van Lanker [4], and DeepSim on B200. Kernels that fail to execute under NVArchSim are excluded, and the remaining kernels are independently sorted in ascending order of relative error for each method.

Figure 8 compares the relative errors of NVArchSim, the analytical model, and DeepSim on B200. Some traces fail under NVArchSim and are excluded; thus, all methods are evaluated on the common subset of kernels that successfully execute under NVArchSim. On the remaining kernels, MAPE is 19.38% for NVArchSim, 21.01% for the analytical model, and 12.30% for DeepSim. Although simulation-based methods can achieve high fidelity when target support is mature, sustaining that fidelity for a new architecture such as B200 requires substantial modeling and calibration effort. Thus, even for NVArchSim, close agreement with emerging hardware remains difficult, highlighting the dependence of simulator accuracy on target-side support maturity. The gap between DeepSim and the analytical model is consistent with the latter's coarser abstraction. While the analytical model supports efficient projection, DeepSim uses a finer-grained intermediate representation based on unit-level *SOL ratios*, which better captures target-side behavior on B200. As a result, DeepSim lowers MAPE by 7.08 and 8.71 percentage points relative to NVArchSim and the analytical model, respectively.

For DeepSim, the profiling report of a kernel on the base GPU is reused across target architectures, so inference cost is constant; the one-time NCU profiling overhead is small and omitted from the runtime comparison. In contrast, NVArchSim requires more than 30 minutes per kernel on average and is more than 1000× slower than both DeepSim and the analytical model. Although the analytical model is about 4× faster than DeepSim, both complete within one second, and their inference cost is independent of kernel execution length, making them far more scalable than simulation-based methods. DeepSim achieves a favorable trade-off: higher accuracy on held-out B200 than methods with stronger target-side assumptions, while retaining low inference cost. It is also designed for the stricter early-stage projection setting, where the target GPU is unavailable and only high-level architectural specifications are accessible.

TABLE IV: Comparison with end-to-end learning methods on the held-out B200 target chip. All methods use the same FTIM features and train/test split as DeepSim.

Model	MAPE	Model	MAPE
Linear Regression	45.45%	MLP	30.70%
Decision Tree	38.52%	Voting Ensemble	43.69%
SVR	55.30%	Stacking Ensemble	47.90%
DeepSim	**12.29%**		

2) Comparing with ML models: Prior data-driven models are discussed qualitatively in Section II-A, since their inputs, datasets, and evaluation targets do not align with our setting. To provide a controlled learning-based comparison, we train representative end-to-end machine learning methods under the same input setting as DeepSim, including linear regression [77], decision tree [78], support vector regression (SVR) [79], and MLP [80]. We further evaluate ensemble learning approaches, including voting and stacking strategies [81], [82]. All models use the same FTIM features and train/test split as DeepSim, and directly predict B200 execution time after hyperparameter tuning. Table IV shows that all end-to-end methods achieve substantially higher MAPE than DeepSim. These results indicate that the improved accuracy of DeepSim does not solely come from the feature set or dataset scale, but benefits from its architecture-aware design and multi-task learning formulation.

F. Component Analysis

1) Function of Workload Classification: Workloads are classified by the unit with the highest predicted *SOL ratio*, representing the dominant performance limiter. Based on the predictions in Section V-B1, 81.56% of kernels are correctly assigned to the same limiting unit as indicated by profiling.

2) Architectural Exploration of Prediction Models: Figure 9 compares architectures for SOL ratio prediction, including Shared Bottom [83], MMoE, and PLE [84]. For MMoE, we evaluate configurations denoted as MMoE-x-y-z, where x is the number of experts, y the expert hidden dimension, and z the tower dimension. Among them, MMoE-6-32-8 achieves the lowest average error (12%) across all SOL ratios and is selected as the final architecture.

Figure 10 compares MAPE across different dual model configurations for four units under two strategies (direct and indirect) and two model choices (MLP and throughput). MLP achieves lower MAPE for SM and L1 than the throughput model, reducing MAPE by 12% and 8% under the direct strategy, respectively. In contrast, the throughput model performs better for L2 and DRAM, reducing MAPE by 15% and 54% compared with MLP. The indirect strategy provides slightly better accuracy than the direct strategy. Replacing the dual model with an analytical-only runtime model increases the overall MAPE from 12.29% to 20.06%, demonstrating the importance of learned runtime modeling even with predicted

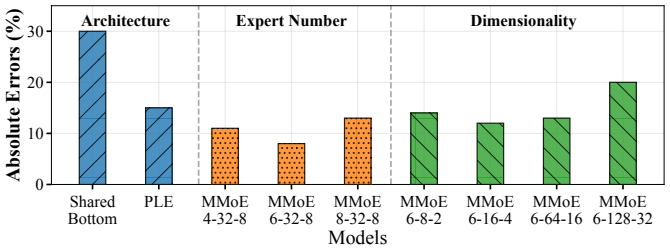


Fig. 9: Absolute errors of *SOL ratio* across model architectures, expert counts, and layer dimensions. In MMoE-x-y-z, x, y, and z denote the number of experts, expert hidden dimension, and tower dimension, respectively.

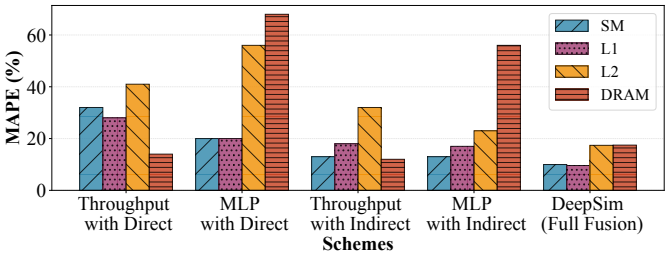


Fig. 10: Comparison of MAPE across different dual model schemes: direct vs. indirect strategies, MLP vs. throughput models, and the Full Fusion scheme in `DeepSim`.

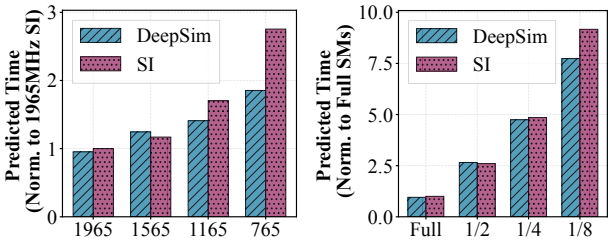
target-side *SOL ratios* (Table III). The Full Fusion scheme in `DeepSim`, which integrates the best-performing configurations across all four units, achieves the lowest MAPE, reaching 9.97%, 9.59%, 17.39%, and 17.48% for SM, L1, L2, and DRAM, respectively.

G. Discussion on Applicability and Limitations

1) Validation under Architectural Variations: We evaluate two configuration changes—GPC frequency and SM count—to verify that `DeepSim` captures performance trends without relying on fixed microarchitectural settings. Figure 11a shows that lower GPC frequencies lead to longer predicted execution times, consistent with expected scaling. Relative errors remain below 17% at 1965, 1565, and 1165 MHz, and increase to 32% at 765 MHz. Figure 11b shows that reducing the number of SMs increases execution time, while relative errors stay within 15%.

2) Sensitivity to Base Chip Selection: `DeepSim` does not require a fixed base chip. On the largest common subset of 98 kernels across all training chips, using H100, GH200, L40, GeForce RTX 5080, and GeForce RTX 5090 as the base chip yields MAPE values of 11.07%, 14.01%, 14.87%, 13.43%, and 17.87% on B200, respectively. These results suggest that base-chip choice affects accuracy but does not prevent cross-chip projection.

3) Applicability across Generations and Vendors: Across NVIDIA GPU generations, `DeepSim` can be applied without architectural modification when the core execution structure



(a) GPC Frequency (MHz) (b) SM Number

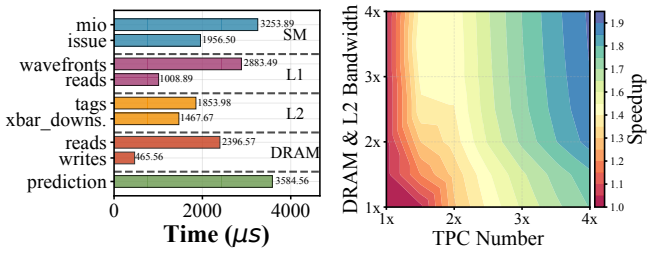
Fig. 11: Predicted execution time normalized to SI (silicon-measured execution time) under different GPC frequencies (1965–765 MHz) and SM counts (Full SMs to 1/8 SMs) on B200. The maximum error is 32% at 765 MHz and 15% at 1/8 SMs, respectively.

remains consistent. Applying the same model to the held-out chip GA100 achieves an MAPE of 16.81%. For GPUs included in the training set, `DeepSim` achieves MAPE values of 10.91% on H100, 8.91% on GH200, 9.01% on L40, 6.42% on GeForce RTX 5080, and 11.50% on GeForce RTX 5090, respectively. When new instructions or hardware units are introduced, `DeepSim` can be adapted with limited additional data if they can be captured by existing or extended FTIM limiters, as demonstrated in Section V-D. Extending `DeepSim` to AMD and Intel GPUs would require mapping vendor-specific profiler metrics from AMD ROCm Compute Profiler [85] or Intel VTune Profiler [86] to FTIM limiters. Differences in hardware hierarchies may introduce additional limiters, such as L3-cache-related limiters for AMD GPUs. Full cross-vendor validation requires corresponding hardware platforms and profiler support.

VI. Case Study

`DeepSim` can be applied across a wide range of domains where performance modeling is critical. To demonstrate its applicability, we present two representative case studies: explainable performance projection for program analysis and chip design space exploration. Figure 12 presents a geophysics kernel from our benchmark suite, which is primarily SM-bound while still placing non-negligible pressure on L2 and DRAM. The predicted *SOL ratios* are 55%, 52%, 39%, and 46% for SM, L1, L2, and DRAM, respectively. Figure 12a shows the top two limiters across the four units. The dominant limiter, *SM:mio*, aligns with the unit exhibiting the maximum *SOL ratio* (SM), and the predicted execution time falls within the expected range—slightly exceeding the limiter's contribution. Figure 12b shows the kernel's performance under a chip-scaling sweep over TPC count and DRAM/L2 bandwidth. Increasing DRAM and L2 bandwidth provides initial performance gains across all TPC configurations, but the improvement quickly saturates because neither DRAM nor L2 is a dominant limiter. In contrast, scaling the number of TPCs raises SM and L1 throughput proportionally and therefore relieves the two units under the highest pressure,

delivering steady speedups that are substantially more effective than DRAM- and L2-side scaling.



(a) (b)

Fig. 12: Geophysics kernel case (mainly SM-bound, with non-negligible L2/DRAM pressure). Baseline: 1× TPC and 1× DRAM/L2 bandwidth. (a) Explainable performance projection under the baseline configuration, showing the top two limiter values for SM, L1, L2, and DRAM. (b) Design space exploration showing performance scaling with TPC number and DRAM & L2 bandwidth, normalized to the baseline.

VII. Conclusion and Future Work

We present `DeepSim`, an explainable data-driven framework for early-stage GPU performance projection on unavailable target chips. By leveraging performance limiters and modeling unit-level execution, imbalance, and frequency effects, `DeepSim` predicts unit-level *SOL ratios* that connect idealized throughput pressure with practical bottleneck behavior. This intermediate representation supports both execution-time prediction and bottleneck-level explainability. Across 284 kernels and 290K samples spanning multiple GPU generations, `DeepSim` achieves strong accuracy on held-out target chips and held-out kernels. On the held-out target chip B200, it achieves an overall MAPE of 12.29%, while the corresponding MAPE on held-out kernels is 14.78%. These results suggest that `DeepSim` provides a practical balance of accuracy, scalability, and interpretability for early-stage GPU projection and design-space exploration.

Future work will extend `DeepSim` in two directions. First, we plan to incorporate a more detailed cache model to better capture cache-induced traffic shifts and memory-side behavior changes across chips. Second, we will extend the framework toward system-level modeling, including host–device communication, to improve projection for workloads with substantial data transfer beyond kernel execution alone.

VIII. Acknowledgment

The authors thank Prof. Xianwei Zhang for valuable comments and suggestions on earlier drafts of this manuscript. The authors used AI-assisted writing tools solely for language editing and proofreading. All technical content, analyses, and conclusions are the original work of the authors.

References

[1] "NVIDIA and Oracle to Build US Department of Energy's Largest AI Supercomputer for Scientific Discovery," https://investor.nvidia.com/news/press-release-details/2025/NVIDIA-and-Oracle-to-Build-US-Department-of-Energys-Largest-AI-Supercomputer-for-Scientific-Discovery/default.aspx.

[2] "Top500 List," https://www.top500.org/lists/top500/2026/06/.

[3] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[4] L. Van Lanker, H. Taboada, E. Brunet, and F. Trahay, "Predicting GPU Kernel's Performance on Upcoming Architectures," in *Euro-Par 2024: Parallel Processing*. Springer Nature Switzerland, 2024, vol. 14801, pp. 77–90. [Online]. Available: https://doi.org/10.1007/978-3-031-69577-3_6

[5] Q. Wang and X. Chu, "GPGPU Performance Estimation With Core and Memory Frequency Scaling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2865–2881, Dec. 2020. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3004623

[6] R. Nath and D. Tullsen, "The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, Dec. 2015, pp. 281–293. [Online]. Available: https://doi.org/10.1145/2830772.2830826

[7] M. Boyer, J. Meng, and K. Kumaran, "Improving GPU Performance Prediction with Data Transfer Modeling," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. IEEE, May 2013, pp. 1097–1106. [Online]. Available: https://doi.org/10.1109/IPDPSW.2013.236

[8] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "GROPHECY: GPU performance projection from CPU code skeletons," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Nov. 2011, pp. 1–11. [Online]. Available: https://doi.org/10.1145/2063384.2063402

[9] E. Konstantinidis and Y. Cotronis, "A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling," *Journal of Parallel and Distributed Computing*, vol. 107, pp. 37–56, Sep. 2017. [Online]. Available: https://doi.org/10.1016/j.jpdc.2017.04.002

[10] C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker, J. Deslippe, and S. Williams, "An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, Nov. 2018, pp. 14–23. [Online]. Available: https://doi.org/10.1109/P3HPC.2018.00005

[11] C. Yang, T. Kurth, and S. Williams, "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 20, p. e5547, Oct. 2020. [Online]. Available: https://doi.org/10.1002/cpe.5547

[12] N. Ding, M. Awan, and S. Williams, "Instruction Roofline: An insightful visual performance model for GPUs," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 20, p. e6591, Sep. 2022. [Online]. Available: https://doi.org/10.1002/cpe.6591

[13] C. Yang, Y. Wang, T. Kurth, S. Farrell, and S. Williams, "Hierarchical Roofline Performance Analysis for Deep Learning Applications," in *Intelligent Computing*. Springer International Publishing, 2021, vol. 284, pp. 473–491. [Online]. Available: https://doi.org/10.1007/978-3-030-80126-7_35

[14] J. Lee, Y. Ha, S. Lee, J. Woo, J. Lee, H. Jang, and Y. Kim, "Gcom: a detailed gpu core model for accurate analytical modeling of modern gpus," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 424–436. [Online]. Available: https://doi.org/10.1145/3470496.3527384

[15] L. Wang, M. Jahre, A. Adileh, and L. Eeckhout, "MDM: The GPU Memory Divergence Model," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, Oct. 2020, pp. 1009–1021. [Online]. Available: https://doi.org/10.1109/MICRO50266.2020.00085

[16] L. Wang, M. Jahre, A. Adileh, Z. Wang, and L. Eeckhout, "Modeling Emerging Memory-Divergent GPU Applications," *IEEE Computer*

Architecture Letters, vol. 18, no. 2, pp. 95–98, Jul. 2019. [Online]. Available: https://doi.org/10.1109/LCA.2019.2923618

[17] S. Lee and W. W. Ro, "Parallel GPU architecture simulation framework exploiting work allocation unit parallelism," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Austin, TX, USA: IEEE, Apr. 2013, pp. 107–117. [Online]. Available: https://doi.org/10.1109/ISPASS.2013.6557151

[18] ——, "Parallel GPU Architecture Simulation Framework Exploiting Architectural-Level Parallelism with Timing Error Prediction," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1253–1265, Apr. 2016. [Online]. Available: https://doi.org/10.1109/TC.2015.2444848

[19] X. Gong, R. Ubal, and D. Kaeli, "Multi2Sim Kepler: A detailed architectural GPU simulator," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Santa Rosa, CA, USA: IEEE, Apr. 2017, pp. 269–278. [Online]. Available: https://doi.org/10.1109/ISPASS.2017.7975298

[20] Y. Arafa, A.-H. Badawy, A. ElWazir, A. Barai, A. Eker, G. Chennupati, N. Santhi, and S. Eidenbenz, "Hybrid, scalable, trace-driven performance modeling of GPGPUs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Nov. 2021, pp. 1–15. [Online]. Available: https://doi.org/10.1145/3458817.3476221

[21] O. Villa, D. Lustig, Z. Yan, E. Bolotin, Y. Fu, N. Chatterjee, N. Jiang, and D. Nellans, "Need for Speed: Experiences Building a Trustworthy System-Level GPU Simulator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, Feb. 2021, pp. 868–880. [Online]. Available: https://doi.org/10.1109/HPCA51647.2021.00077

[22] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, May 2020, pp. 473–486. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00047

[23] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, Apr. 2009, pp. 163–174. [Online]. Available: https://doi.org/10.1109/ISPASS.2009.4919648

[24] X. Wang, K. Huang, A. Knoll, and X. Qian, "A Hybrid Framework for Fast and Accurate GPU Performance Estimation through Source-Level Analysis and Trace-Based Simulation," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2019, pp. 506–518. [Online]. Available: https://doi.org/10.1109/HPCA.2019.00062

[25] J. Lowe-Power *et al.*, "The gem5 Simulator: Version 20.0+," Sep. 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2007.03152

[26] P. Gera, H. Kim, H. Kim, S. Hong, V. George, and C.-K. Luk, "Performance Characterisation and Simulation of Intel's Integrated GPU Architecture," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2018, pp. 139–148. [Online]. Available: https://doi.org/10.1109/ISPASS.2018.00027

[27] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A simulation framework for CPU-GPU computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, Sep. 2012, pp. 335–344. [Online]. Available: https://doi.org/10.1145/2370816.2370865

[28] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao, H. Barclay, A. K. Ziabari, Z. Chen, R. Ubal, J. L. Abellán, J. Kim, A. Joshi, and D. Kaeli, "MGPUSim: Enabling multi-GPU performance modeling and optimization," in *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, Jun. 2019, pp. 197–209. [Online]. Available: https://doi.org/10.1145/3307650.3322230

[29] H. Kim, J. Lee, N. B. Lakshminarayana, J. Sim, J. Lim, and T. Pho, "Macsim: A cpu-gpu heterogeneous simulation framework user guide," *Georgia Institute of Technology*, pp. 1–57, 2012.

[30] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers, and T. M. Aamodt, "Analyzing Machine Learning Workloads Using a Detailed GPU Simulator," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2019, pp. 151–152. [Online]. Available: https://doi.org/10.1109/ISPASS.2019.00028

[31] H. SeyyedAghaei, M. Naderan-Tahan, and L. Eeckhout, "GPU Scale-Model Simulation," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Edinburgh, United Kingdom: IEEE, Mar. 2024, pp. 1125–1140. [Online]. Available: https://doi.org/10.1109/HPCA57654.2024.00088

[32] J. Yang, M. Wen, D. Chen, Z. Chen, Z. Xue, Y. Li, J. Shen, and Y. Shi, "HyFiSS: A Hybrid Fidelity Stall-Aware Simulator for GPGPUs," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Austin, TX, USA: IEEE, Nov. 2024, pp. 168–185. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00022

[33] K. O'Neal, P. Brisk, E. Shriver, and M. Kishinevsky, "HALWPE: Hardware-Assisted Light Weight Performance Estimation for GPUs," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, Jun. 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1145/3061639.3062257

[34] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2015, pp. 564–576. [Online]. Available: https://doi.org/10.1109/HPCA.2015.7056063

[35] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, Dec. 2015, pp. 725–737. [Online]. Available: https://doi.org/10.1145/2830772.2830780

[36] R. Zheng, Q. Hu, and H. Jin, "GPUPerfML: A Performance Analytical Model Based on Decision Tree for GPU Architectures," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, Jun. 2018, pp. 602–609. [Online]. Available: https://doi.org/10.1109/HPCC/SmartCity/DSS.2018.00110

[37] T. T. Dao, J. Kim, S. Seo, B. Egger, and J. Lee, "A Performance Model for GPUs with Caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 1800–1813, Jul. 2015. [Online]. Available: https://doi.org/10.1109/TPDS.2014.2333526

[38] "NVIDIA Nsight Compute Profiling Guide," https://docs.nvidia.com/nsight-compute/2025.3/ProfilingGuide/index.html.

[39] H. Cha, S. Lee, J. Lee, Y. Ha, J. Kim, and Y. Kim, "GCStack+GCScaler: Fast and Accurate GPU Performance Analyses Using Fine-Grained Stall Cycle Accounting and Interval Analysis," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. ACM, Jun. 2025, pp. 1509–1523. [Online]. Available: https://doi.org/10.1145/3695053.3731068

[40] J.-C. Huang, J. H. Lee, H. Kim, and H.-H. S. Lee, "GPUMech: GPU Performance Modeling Technique Based on Interval Analysis," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, Dec. 2014, pp. 268–279. [Online]. Available: https://doi.org/10.1109/MICRO.2014.59

[41] Y. Arafa, A.-H. A. Badawy, G. Chennupati, N. Santhi, and S. Eidenbenz, "PPT-GPU: Scalable GPU Performance Modeling," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 55–58, Jan. 2019. [Online]. Available: https://doi.org/10.1109/LCA.2019.2904497

[42] C. Lehnert, R. Berrendorf, J. P. Ecker, and F. Mannuss, "Performance Prediction and Ranking of SpMV Kernels on GPU Architectures," in *Euro-Par 2016: Parallel Processing*. Springer International Publishing, 2016, vol. 9833, pp. 90–102. [Online]. Available: https://doi.org/10.1007/978-3-319-43659-3_7

[43] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based GPU design space exploration," in *2012 IEEE International Symposium on Performance Analysis of Systems & Software*. New Brunswick, NJ, USA: IEEE, Apr. 2012, pp. 2–13. [Online]. Available: https://doi.org/10.1109/ISPASS.2012.6189201

[44] Y. Li, Y. Sun, and A. Jog, "Path forward beyond simulators: Fast and accurate gpu execution time prediction for dnn workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 380–394. [Online]. Available: https://doi.org/10.1145/3613424.3614277

[45] A. Benatia, W. Ji, Y. Wang, and F. Shi, "Machine Learning Approach for the Predicting Performance of SpMV on GPU," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. Wuhan, China: IEEE, Dec. 2016, pp. 894–901. [Online]. Available: https://doi.org/10.1109/ICPADS.2016.0120

[46] W. Jia, K. A. Shaw, and M. Martonosi, "Starchart: Hardware and Software Optimization Using Recursive Partitioning Regression Trees,"

in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*. Edinburgh: IEEE, Oct. 2013, pp. 331–340. [Online]. Available: https://doi.org/10.1109/PACT.2013.6618822

[47] B. Johnston, G. Falzon, and J. Milthorpe, "OpenCL Performance Prediction using Architecture-Independent Features," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, Jul. 2018, pp. 561–569. [Online]. Available: https://doi.org/10.1109/HPCS.2018.00095

[48] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, "A Simple Model for Portable and Fast Prediction of Execution Time and Power Consumption of GPU Kernels," *ACM Transactions on Architecture and Code Optimization*, vol. 18, no. 1, pp. 1–25, Mar. 2021. [Online]. Available: https://doi.org/10.1145/3431731

[49] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, May 2013, pp. 673–686. [Online]. Available: https://doi.org/10.1109/IPDPS.2013.73

[50] M. Amaris, R. Y. De Camargo, M. Dyab, A. Goldman, and D. Trystram, "A comparison of GPU execution time prediction using machine learning and analytical modeling," in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. IEEE, Oct. 2016, pp. 326–333. [Online]. Available: https://doi.org/10.1109/NCA.2016.7778637

[51] S. Lee, A. Phanishayee, and D. Mahajan, "Forecasting GPU Performance for Deep Learning Training and Inference," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. ACM, Mar. 2025, pp. 493–508. [Online]. Available: https://doi.org/10.1145/3669940.3707265

[52] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, "GPU Static Modeling Using PTX and Deep Structured Learning," *IEEE Access*, vol. 7, pp. 159 150–159 161, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2951218

[53] S. Salaria, A. Drozd, A. Podobas, and S. Matsuoka, "Learning Neural Representations for Predicting GPU Performance," in *High Performance Computing*. Springer International Publishing, 2019, vol. 11501, pp. 40–58. [Online]. Available: https://doi.org/10.1007/978-3-030-20656-7_3

[54] K. N. M. Nguyen, H. D. N. Do, H. T. Le, and T. T. Dao, "LLMPerf: GPU Performance Modeling meets Large Language Models," Mar. 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2503.11244

[55] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks," May 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1808.07412

[56] L. Li, T. Flynn, and A. Hoisie, "Learning Generalizable Program and Architecture Representations for Performance Modeling," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2024, pp. 1–15. [Online]. Available: https://doi.org/10.1109/SC41406.2024.00072

[57] "Nsight Compute CLI," https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html.

[58] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1930–1939. [Online]. Available: https://doi.org/10.1145/3219819.3220007

[59] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, and R. Strzodka, "AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods," *SIAM Journal on Scientific Computing*, vol. 37, no. 5, pp. S602–S626, 2015. [Online]. Available: https://doi.org/10.1137/140980260

[60] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1–2, pp. 19–25, 2015. [Online]. Available: https://doi.org/10.1016/j.softx.2015.06.001

[61] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS: A flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications*, vol. 271, p. 108171, 2022. [Online]. Available: https://doi.org/10.1016/j.cpc.2021.108171

[62] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, and J. Tromp, "Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes," *Geophysical Journal International*, vol. 186, no. 2, pp. 721–739, 2011. [Online]. Available: https://doi.org/10.1111/j.1365-246X.2011.05044.x

[63] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, "Solving lattice QCD systems of equations using mixed precision solvers on GPUs," *Computer Physics Communications*, vol. 181, no. 9, pp. 1517–1528, 2010. [Online]. Available: https://doi.org/10.1016/j.cpc.2010.05.002

[64] *TPC Benchmark H (TPC-H), Standard Specification*, Transaction Processing Performance Council. [Online]. Available: https://www.tpc.org/tpch/

[65] J. Hutter, M. Iannuzzi, F. Schiffmann, and J. VandeVondele, "CP2K: Atomistic simulations of condensed matter systems," *WIREs Computational Molecular Science*, vol. 4, no. 1, pp. 15–25, 2014. [Online]. Available: https://doi.org/10.1002/wcms.1159

[66] "NVIDIA Ampere GPU Architecture Tuning Guide," https://docs.nvidia.com/cuda/ampere-tuning-guide/index.html.

[67] "NVIDIA ADA GPU ARCHITECTURE," https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf.

[68] "NVIDIA H100 Tensor Core GPU Architecture," https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c.

[69] "NVIDIA GH200 Grace Hopper Superchip," https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip.

[70] "NVIDIA RTX BLACKWELL GPU ARCHITECTURE," https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf.

[71] "NVIDIA DGX B200," https://docs.nvidia.com/dgx/dgxb200-user-guide/dgxb200-user-guide.pdf.

[72] "NVIDIA Multi-Instance GPU," https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[73] "NVIDIA Nsight Compute," https://developer.nvidia.com/nsight-compute.

[74] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[75] "DeepSeek-V3.1 Configuration," https://huggingface.co/deepseek-ai/DeepSeek-V3.1/blob/main/configuration_deepseek.py.

[76] "NVIDIA CUTLASS Documentation," https://docs.nvidia.com/cutlass/index.html.

[77] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009. [Online]. Available: https://doi.org/10.1007/978-0-387-84858-7

[78] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth International Group, 1984. [Online]. Available: https://www.taylorfrancis.com/books/mono/10.1201/9781315139470/classification-regression-trees-leo-breiman

[79] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[80] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[81] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996. [Online]. Available: https://doi.org/10.1007/BF00058655

[82] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992. [Online]. Available: https://doi.org/10.1016/S0893-6080(05)80023-1

[83] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.05098

[84] H. Tang, J. Liu, M. Zhao, and X. Gong, "Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations," in *Proceedings of the 14th ACM Conference on Recommender Systems*, ser. RecSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 269–278. [Online]. Available: https://doi.org/10.1145/3383313.3412236

[85] AMD, “Rocm compute profiler,” https://github.com/ROCm/rocprofiler-compute.
[86] Intel Corporation, “Intel vtune profiler,” https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html.