A Survey of High-Level Modeling and Simulation Methods for Modern Machine Learning Workloads

MICRO 2026 Submission – Confidential Draft – Do NOT Distribute!!

Abstract

As machine learning workloads grow in scale and complexity, architects and system designers need fast, accurate methods to predict their performance across diverse hardware platforms. This survey provides a comprehensive analysis of modeling and simulation methods for predicting the performance of ML workloads, covering analytical models, cycle-accurate simulators, trace-driven approaches, and ML-augmented hybrid techniques. We survey over 50 tools and methods from architecture and systems venues published between 2016–2026, spanning DNN accelerator modeling (Timeloop, MAESTRO, Sparseloop), GPU simulation (GPGPU-Sim, Accel-Sim, NeuSight), distributed training simulation (ASTRA-sim, Lumos, SimAI), and LLM inference serving (VIDUR, Frontier, AMALI). We organize the literature along two primary dimensions—methodology type (analytical, simulation, ML-augmented, hybrid) and target platform (accelerators, GPUs, distributed systems, edge devices)—while additionally characterizing tools by workload coverage, prediction targets, and independently verified accuracy. Our analysis reveals that hybrid approaches combining analytical structure with learned components achieve the best accuracy-speed trade-offs, while pure analytical models offer superior interpretability for design space exploration. We conduct hands-on reproducibility evaluations and report independently measured accuracy, finding significant gaps between paper-reported and independently verified numbers. We identify key open challenges including cross-workload generalization beyond CNNs, composition of kernel-level predictions to end-to-end accuracy, and support for emerging architectures. This survey provides practitioners guidance for selecting appropriate modeling tools and researchers a roadmap for advancing ML workload performance prediction.

Keywords

performance modeling, machine learning workloads, simulation, computer architecture, design space exploration, survey

1 Introduction

Machine learning workloads—spanning training and inference for CNNs, transformers, mixture-of-experts models, and graph neural networks—have become the dominant consumers of compute across datacenters and edge devices. Architects and system designers need fast, accurate performance predictions to navigate vast design spaces, select parallelization strategies, provision serving infrastructure, and optimize hardware-software co-design. Yet ML workloads pose unique modeling challenges: they exhibit diverse computational patterns (dense matrix operations in attention layers, sparse accesses in GNNs, communication-bound collective operations in distributed training) across an increasingly heterogeneous hardware landscape of GPUs, TPUs, custom accelerators, and multi-device clusters.

MICRO 2026, Austin, TX, USA
2026. ACM ISBN 978-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

A rich ecosystem of modeling and simulation tools has emerged to address these challenges, spanning a methodological spectrum from analytical models to cycle-accurate simulators to ML-augmented hybrid approaches. Analytical frameworks like Timeloop [32] and MAESTRO [23] model DNN accelerator performance through closed-form data movement analysis, achieving 5–10% accuracy at microsecond evaluation speed. Cycle-accurate simulators like GPGPU-Sim [3] and Accel-Sim [19] provide detailed GPU modeling but require hours per workload. Trace-driven simulators like ASTRA-sim [42] and VIDUR [2] target distributed training and LLM serving at system scale. ML-augmented approaches like NeuSight [26] learn performance functions from profiling data, achieving 2.3% error on GPU kernel prediction. Each methodology occupies a distinct point in the accuracy-speed-generality trade-off space.

Despite this rich tool landscape, no comprehensive survey organizes these methods from the perspective of the ML workload practitioner. Existing surveys focus on ML *techniques* for performance modeling [39] or on specific hardware targets [32], leaving practitioners without guidance on which tools suit their needs across the full modeling spectrum. This survey fills that gap.

We make the following contributions:

- A **methodology-centric taxonomy** organizing tools along two primary dimensions: methodology type (analytical, simulation, ML-augmented, hybrid) and target platform (DNN accelerators, GPUs, distributed systems, edge devices), with additional characterization by workload coverage, prediction targets, and verified accuracy.
- A **systematic survey** of over 50 modeling tools and methods from architecture venues (MICRO, ISCA, HPCA, ASPLOS) and systems venues (MLSys, OSDI, NSDI) published between 2016–2026, using documented selection criteria.
- A **comparative analysis** examining trade-offs between accuracy, speed, generalization, and interpretability, with independently measured accuracy where feasible rather than relying solely on paper-reported numbers.
- **Hands-on reproducibility evaluations** of representative tools with a 10-point rubric, and identification of **open challenges** including the CNN-to-transformer generalization gap, kernel-to-end-to-end error composition, and emerging accelerator support.

The remainder of this paper is organized as follows. Section 2 describes our survey methodology. Section 3 provides background on ML workload characteristics and modeling fundamentals. Section 4 presents our classification taxonomy. Section 5 surveys approaches organized by target platform. Section 6 offers comparative analysis across key dimensions. Section 7 discusses open challenges and future directions. Section 8 presents hands-on reproducibility evaluations. Section 9 concludes.

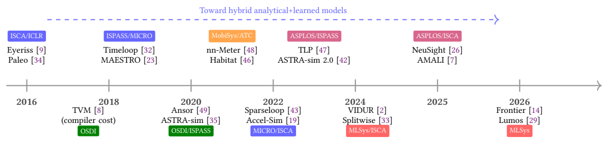

Figure 1: Evolution of performance modeling tools for ML workloads (2016–2026). Early analytical frameworks (Eyeriss, Paleo) gave way to systematic accelerator modeling (Timeloop, MAESTRO) and distributed training simulation (ASTRA-sim). ML-augmented approaches (TVM, Habitat, NeuSight) learn performance functions from data. Recent work targets LLM-specific modeling (VIDUR, AMALI, Frontier) and large-scale training prediction (Lumos).

Figure 1 illustrates the evolution of performance modeling tools for ML workloads, from early analytical frameworks through simulators to modern hybrid approaches.

2 Survey Methodology

We follow a systematic methodology for identifying, selecting, and classifying papers in this survey.

Search strategy. We searched ACM Digital Library, IEEE Xplore, Semantic Scholar, and arXiv using terms including "performance modeling DNN," "DNN accelerator simulator," "LLM inference prediction," "distributed training simulation," "neural network latency estimation," and "ML workload performance." We additionally performed backward/forward citation tracking from seminal works (Timeloop, ASTRA-sim, NeuSight) and monitored proceedings of target venues.

Target venues. Architecture: MICRO, ISCA, HPCA, ASPLOS. Systems: MLSys, OSDI, SOSP, NSDI. Related: NeurIPS, ICML, MobiSys, DAC, ISPASS.

Inclusion criteria. Papers must (1) propose or evaluate a tool or method for predicting performance of ML workloads (training or inference), (2) target at least one hardware platform (GPU, accelerator, distributed system, or edge device), and (3) include quantitative evaluation of prediction accuracy or modeling fidelity.

Exclusion criteria. We exclude (1) papers using ML for non-performance tasks (e.g., power estimation without latency), (2) papers modeling general-purpose (non-ML) workloads exclusively, and (3) papers without quantitative evaluation.

Selection process. Our initial search yielded 287 candidate papers. After title/abstract screening against inclusion criteria, 118 remained. Full-text review reduced the set to 53 papers that met all criteria. We additionally include 12 foundational works (gem5, roofline model, DRAMSim, etc.) as context for understanding the modeling landscape.

Time period. We cover papers published between 2016–2026, with foundational works from earlier years included for context.

Classification. We classify each paper along two primary dimensions: *methodology type* (analytical, cycle-accurate simulation, trace-driven simulation, ML-augmented, or hybrid) and *target platform* (DNN accelerator, GPU, distributed system, edge device, or CPU). Secondary dimensions include workload coverage, prediction targets, and reported accuracy metrics.

3 Background

This section provides background on the characteristics of ML workloads that make performance modeling challenging, and reviews the fundamental approaches used to model them.

3.1 ML Workload Characteristics

ML workloads present unique performance modeling challenges compared to general-purpose programs.

Computational structure. ML workloads are composed of well-defined operators (convolutions, matrix multiplications, attention layers, normalization) with statically known shapes and data types. This regularity enables analytical modeling of compute and data movement, unlike branch-heavy general-purpose code. However, modern architectures like mixture-of-experts (MoE) and dynamic inference introduce input-dependent control flow that complicates static analysis.

Memory hierarchy sensitivity. DNN accelerators employ specialized memory hierarchies with explicit data orchestration. The mapping of tensor operations to hardware (dataflow, tiling, loop ordering) critically determines performance. For LLM inference, KV cache management dominates memory behavior, with cache sizes scaling linearly with sequence length and batch size [24].

Scale and distribution. Large model training distributes computation across thousands of GPUs using data, tensor, pipeline, and expert parallelism [?]. Performance depends on the interplay between compute, memory bandwidth, and network communication—requiring system-level modeling beyond single-device prediction.

Distinct inference phases. LLM inference exhibits qualitatively different phases: prefill (compute-bound, processing the full prompt) and decode (memory-bound, generating tokens autoregressively) [33]. Effective modeling must capture both phases and their interaction under batched serving [1, 45].

3.2 Modeling Methodologies

We classify modeling approaches into four categories that form the primary axis of our taxonomy.

Analytical models express performance as closed-form functions of workload and hardware parameters. The roofline model [41] bounds throughput by $P = \min(\pi, \beta \cdot I)$, where π is peak compute, β is memory bandwidth, and I is operational intensity. For DNN accelerators, Timeloop [32] analytically computes data movement costs across memory hierarchies for any valid mapping. Analytical models provide microsecond evaluation and full interpretability, but require manual derivation per architecture and struggle with dynamic microarchitectural effects.

Cycle-accurate simulators model hardware at the register-transfer level. gem5 [5] (CPUs), GPGPU-Sim [3] (GPUs), and Accel-Sim [19] (modern GPUs) achieve detailed accuracy but suffer 1000–10000× slowdown, making them impractical for full ML workload evaluation. Sampling techniques (SimPoint [37], SMARTS [44]) reduce simulation time but were designed for general-purpose workloads and may not capture ML-specific patterns.

Trace-driven simulation uses execution traces as input rather than full binary execution, enabling faster evaluation. ASTRA-sim [42] models distributed training using Chakra execution traces [38] with pluggable compute, memory, and network backends. VIDUR [2]

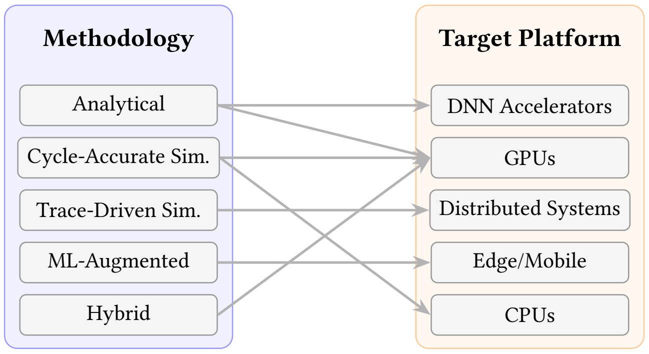

Figure 2: Two-dimensional taxonomy for ML workload performance modeling. The primary axis is methodology type (how performance is predicted); the secondary axis is target platform. Arrows show dominant pairings: analytical models for accelerators, cycle-accurate simulation for GPUs/CPUs, trace-driven simulation for distributed systems, and ML-augmented approaches for edge devices.

provides discrete-event simulation for LLM serving using kernel-level profiles. This approach trades some fidelity for orders-of-magnitude speedup over cycle-accurate simulation.

ML-augmented approaches learn performance functions from profiling data. These range from simple models (random forests in nn-Meter [48], XGBoost in TVM [8]) to deep learning (NeuSight [26]) and meta-learning (HELP [25]). ML-augmented approaches can capture complex non-linear relationships that elude analytical treatment, but require training data and may not generalize beyond their training distribution.

3.3 Problem Formulation

Performance modeling maps workload $\mathcal{W}$ and hardware $\mathcal{H}$ to a performance metric y: $\hat{y} = f(\mathcal{W}, \mathcal{H}; \theta)$. Workloads are represented at operator level (layer parameters), graph level (computation graphs), IR level (compiler representations), or trace level (recorded runtime behavior). Hardware is characterized by specifications, performance counters, or learned embeddings.

Prediction targets include latency (execution time), throughput (samples/second), energy (Joules per inference), and memory footprint. Multi-objective formulations enable Pareto-optimal design selection.

Accuracy metrics vary across the literature: MAPE (scale-invariant relative error), RMSE (penalizes large deviations), and rank correlation (Kendall's τ) for design space ordering. Direct comparison across papers is limited by differences in benchmarks, hardware targets, and evaluation protocols—a challenge we discuss in Section 6.

4 Taxonomy

We organize the literature along two primary dimensions: *methodology type* and *target platform*. Figure 2 illustrates how these dimensions intersect. This methodology-centric taxonomy helps practitioners select appropriate tools for their use case and helps researchers identify gaps in the current landscape.

4.1 By Methodology Type

The choice of methodology reflects fundamental trade-offs between accuracy, evaluation speed, generality, and interpretability.

4.1.1 Analytical Models. Analytical models express performance as closed-form functions of workload and hardware parameters. For DNN accelerators, Timeloop [32] models data movement across memory hierarchies for any valid loop-nest mapping, achieving 5–10% accuracy versus RTL at 2000× speedup. MAESTRO [23] provides data-centric dataflow analysis using intuitive directives. Sparseloop [43] extends to sparse tensor operations. Paleo [34] pioneered layer-wise analytical modeling for DNNs, decomposing networks into compute and communication components for distributed training prediction. AMALI [7] targets LLM inference on GPUs through improved memory hierarchy modeling.

Analytical models provide microsecond evaluation, full interpretability, and "what-if" design analysis. Their limitation is that they require manual derivation per architecture and may miss complex dynamic effects (e.g., memory contention, scheduling variability). AMALI's 23.6% MAPE illustrates the challenge: reducing GPU LLM inference modeling error from 127% to 24% through improved memory hierarchy treatment, but significant residual error remains from unmodeled dynamic effects—this represents the fundamental difficulty of analytical approaches for complex workloads, not a quality issue [7].

4.1.2 Cycle-Accurate Simulation. Cycle-accurate simulators model hardware at register-transfer level, providing the highest fidelity. gem5 [5] (CPUs), GPGPU-Sim [3] (GPUs), and Accel-Sim [19] (modern NVIDIA GPUs, SASS-level trace-driven) achieve 0.90–0.97 IPC correlation. PyTorchSim [20] integrates PyTorch 2 with NPU simulation supporting custom RISC-V ISA and systolic arrays.

The primary limitation is speed: simulating a single ResNet-50 inference may require hours, making these tools impractical for design space exploration of ML workloads. Simulation sampling techniques (SimPoint [37], SMARTS [44], LoopPoint [36]) accelerate general-purpose workload simulation but are not specifically validated for ML workload patterns. Recent work on dissecting modern GPU cores [?] has improved Accel-Sim's accuracy to 13.98% MAPE by reverse-engineering undocumented microarchitectural details.

4.1.3 Trace-Driven Simulation. Trace-driven approaches use recorded execution traces rather than full binary execution, enabling system-level modeling at practical speeds. ASTRA-sim [42] models distributed training end-to-end using Chakra execution traces [38], with pluggable compute, memory, and network backends, achieving 5–15% accuracy versus real clusters. Echo [6] simulates distributed training at scale using analytical compute models with network simulation. Lumos [29] targets LLM training performance through trace-driven modeling, achieving 3.3% error on H100 GPUs.

For LLM inference serving, VIDUR [2] provides discrete-event simulation capturing prefill/decode phases, KV cache management, and request scheduling (Orca [45], Sarathi [1] strategies) with <5% error. Frontier [14] extends to MoE and disaggregated inference with stage-centric simulation. SimAI [?] provides full-stack LLM training simulation achieving 98.1% alignment with production results at Alibaba Cloud scale.

These tools occupy a practical middle ground: fast enough for design exploration, detailed enough to capture system-level interactions that analytical models miss. Note that some tools in this category use ML internally (e.g., VIDUR uses random forests for kernel latency prediction), blurring the boundary with hybrid approaches.

4.1.4 ML-Augmented Models. ML-augmented approaches learn performance functions entirely from profiling data, without embedding analytical domain knowledge. nn-Meter [48] uses random forest ensembles with kernel-level feature engineering for edge device latency prediction. LitePred [12] scales to 85 edge platforms using VAE-based intelligent sampling and transfer learning. HELP [25] formulates cross-hardware prediction as meta-learning, achieving adaptation with just 10 samples on new devices. TVM [8] and Ansor [49] use XGBoost/MLP cost models to guide compiler autotuning, with the TenSet dataset [50] (52M records) enabling pre-trained models.

ML-augmented approaches excel when sufficient profiling data is available and the training distribution matches deployment conditions. However, they may fail silently outside their training distribution (e.g., CNN-trained models applied to transformers) and provide limited interpretability for design insight. nn-Meter's paper-reported <1% MAPE cannot be independently verified, as the tool's pre-trained predictors fail with current scikit-learn versions due to pickle serialization changes—a cautionary example of how ML-augmented approaches can become irreproducible.

4.1.5 Hybrid Analytical+ML Models. Hybrid approaches combine analytical structure with learned components, achieving both interpretability and high accuracy. The analytical component provides a physics-based prior; the ML component learns residual corrections.

NeuSight [26] uses tile-based prediction mirroring CUDA's execution model with MLP prediction heads, achieving 2.3% error on GPT-3 inference across H100, A100, and V100 GPUs. Concorde [31] fuses compositional analytical models with learned corrections for CPU performance, achieving 2% CPI error at five orders of magnitude faster than gem5. Habitat [46] decomposes execution into analytically-modeled compute and memory components that scale with hardware parameters. ArchGym [22] connects ML optimization algorithms to analytical simulators for design space exploration.

The latency predictor study [11] demonstrates that hybrid approaches with transfer learning achieve 22.5% average improvement over baselines. Note that accuracy comparisons between hybrid and pure approaches require care: ArchGym's reported 0.61% RMSE measures surrogate-vs-simulator fidelity, not real hardware accuracy, and should not be directly compared with NeuSight's 2.3% MAPE against measured hardware [22].

4.2 By Target Platform

The target platform determines what performance effects must be modeled and constrains which methodologies are applicable.

DNN Accelerators (systolic arrays, dataflow architectures) are best served by analytical models (Timeloop, MAESTRO, Sparseloop) due to their regular, statically analyzable memory hierarchies and explicit dataflow control.

GPUs span the full methodology spectrum, from cycle-accurate (GPGPU-Sim, Accel-Sim) through analytical (AMALI, roofline [17, 41]) to hybrid (NeuSight, Habitat), reflecting the complexity of SIMT execution, warp scheduling, and memory coalescing.

Distributed systems are primarily served by trace-driven simulation (ASTRA-sim, VIDUR, Lumos, SimAI, Frontier) because system-level interactions (collective communication, pipeline parallelism, scheduling) cannot be captured by single-device models.

Edge/mobile devices are dominated by ML-augmented approaches (nn-Meter, LitePred, HELP) because the diversity of edge hardware makes per-device analytical modeling impractical.

CPUs for ML workloads are less studied because most ML training and inference runs on GPUs/accelerators. Concorde and GRANITE [39] target CPU performance but focus on general-purpose workloads rather than ML-specific patterns.

5 Survey of Approaches

This section surveys performance modeling tools for ML workloads, organized by target platform. For each platform, we examine the modeling challenges, describe the available tools across methodology types, and critically analyze their strengths and limitations. Table 1 provides a comprehensive comparison.

5.1 DNN Accelerator Modeling

DNN accelerators employ specialized dataflows and memory hierarchies optimized for tensor operations. The regularity of DNN computations makes this domain particularly amenable to analytical modeling.

Analytical frameworks dominate accelerator modeling. Timeloop [32] analytically computes data reuse, latency, and energy from loop-nest representations, achieving 5–10% accuracy versus RTL simulation at 2000× speedup. It provides reference outputs for standard accelerator designs (Eyeriss [9], Simba) with deterministic results—a key reproducibility strength. MAESTRO [23] offers data-centric dataflow directives that simplify specification but is less precise than Timeloop for detailed energy modeling. Sparseloop [43] extends to sparse tensor operations, critical for efficient transformer inference where attention matrices exhibit structured sparsity.

Simulation approaches. PyTorchSim [20] integrates PyTorch 2 with NPU simulation supporting custom RISC-V ISA and systolic arrays, bridging the gap between ML frameworks and hardware simulation.

ML-augmented design. ArchGym [22] connects ML optimization algorithms to analytical simulators for design space exploration. Its reported 0.61% RMSE measures how faithfully the ML surrogate reproduces the simulator's predictions—not accuracy against real hardware. This distinction matters: surrogate fidelity enables fast DSE but does not validate the underlying simulator's accuracy.

Emerging accelerators. Processing-in-memory (PIM) architectures present fundamentally different modeling challenges. uPIMulator [?] provides cycle-accurate PIM simulation for UPMEM devices. AttAcc [?] and NeuPIMs [?] target PIM-based attention acceleration for transformers, while PAISE [?] addresses PIM-accelerated LLM inference scheduling. These tools remain in early stages compared to the mature DNN accelerator modeling ecosystem.

Table 1: Summary of surveyed performance modeling tools for ML workloads, organized by target platform. Methodology: A=Analytical, S=Simulation, T=Trace-driven, M=ML-augmented, H=Hybrid. *Accuracy measures surrogate-vs-simulator fidelity, not real hardware error. †Reported accuracy unverifiable due to reproducibility issues. ‡No accuracy baseline against real hardware reported.

Tool	Platform	Method	Target	Accuracy	Speed	Key Capability
DNN Accelerator Modeling						
Timeloop [32]	NPU	A	Latency/Energy	5–10%	μs	Loop-nest DSE
MAESTRO [23]	NPU	A	Latency/Energy	5–15%	μs	Data-centric directives
Sparseloop [43]	NPU	A	Sparse tensors	5–10%	μs	Compression modeling
PyTorchSim [20]	NPU	S	Cycle-accurate	N/A‡	Hours	PyTorch 2 integration
ArchGym [22]	Multi	H	Multi-objective	0.61%*	ms	ML-aided DSE
GPU Performance Modeling						
Accel-Sim [19]	GPU	S	Cycle-accurate	10–20%	Hours	SASS trace-driven
GPGPU-Sim [3]	GPU	S	Cycle-accurate	10–20%	Hours	CUDA workloads
AMALI [7]	GPU	A	LLM inference	23.6%	ms	Memory hierarchy
NeuSight [26]	GPU	H	Kernel/E2E latency	2.3%	ms	Tile-based prediction
Habitat [46]	GPU	H	Training time	11.8%	Per-kernel	Wave scaling
Distributed Training and LLM Serving						
ASTRA-sim [42]	Distributed	T	Training time	5–15%	Minutes	Collective modeling
SimAI [?]	Distributed	T	Training time	1.9%	Minutes	Full-stack simulation
Lumos [29]	Distributed	T	LLM training	3.3%	Minutes	H100 training
VIDUR [2]	GPU cluster	T	LLM serving	<5%	Seconds	Prefill/decode phases
Frontier [14]	Distributed	T	MoE inference	—	Minutes	Stage-centric sim.
TrioSim [27]	Multi-GPU	T	DNN training	N/A‡	Minutes	Lightweight multi-GPU
Edge Device Modeling						
nn-Meter [48]	Edge	M	Latency	<1%†	ms	Kernel detection
LitePred [12]	Edge	M	Latency	0.7%	ms	85-platform transfer
HELP [25]	Multi	M	Latency	1.9%	ms	10-sample adaptation
Compiler Cost Models						
TVM [8]	GPU	M	Schedule perf.	∼15%	ms	Autotuning guidance
Ansor [49]	GPU	M	Schedule perf.	∼15%	ms	Program sampling
TLP [47]	GPU	M	Tensor program	<10%	ms	Transformer cost model

5.2 GPU Performance Modeling

GPUs dominate ML training and inference, making accurate GPU performance prediction critical. GPU modeling must account for SIMT execution, warp scheduling, memory coalescing, and workload-dependent occupancy effects.

Cycle-accurate simulation. GPGPU-Sim [3] and Accel-Sim [19] achieve 0.90–0.97 IPC correlation through detailed microarchitectural modeling. Recent work reverse-engineering modern GPU cores [?] has improved Accel-Sim to 13.98% MAPE by modeling previously undocumented features. However, 1000–10000× slowdown makes these tools impractical for full ML workloads at production scale.

Analytical models. The roofline model [41] provides a useful upper bound but misses occupancy and memory hierarchy effects. Roofline-LLM [17] extends roofline analysis to LLM inference. AMALI [7] reduces GPU LLM inference MAPE from 127% (prior analytical baselines) to 23.6% through improved memory hierarchy modeling. The residual 23.6% error reflects the fundamental difficulty of analytically modeling GPU dynamic behavior (warp scheduling, L2 cache contention, bank conflicts) rather than a quality limitation.

Hybrid learned models. NeuSight [26] introduces tile-based prediction that mirrors CUDA's execution model, achieving 2.3% MAPE on GPT-3 inference across H100, A100, and V100 GPUs. Habitat [46] decomposes execution into analytically-modeled compute and memory components using wave scaling analysis, achieving 11.8% error across GPU generations. Note that direct comparison between NeuSight and Habitat requires caution: NeuSight evaluates on 2023–2025 hardware (H100) with LLM workloads, while Habitat was designed for earlier GPUs with CNN/RNN workloads—the reported "50× improvement" reflects different evaluation conditions rather than purely methodological advances.

LLM-specific modeling. LLM execution exhibits qualitatively different prefill (compute-bound) and decode (memory-bound) phases [33, 51]. VIDUR [2] provides discrete-event simulation for LLM serving systems, capturing request scheduling strategies (Orca [45], Sarathi [1]) with <5% error. LIFE [13] offers hardware-agnostic analytical LLM inference modeling. HERMES [4] targets heterogeneous multi-stage inference pipelines. Emerging work uses LLMs themselves for GPU kernel performance prediction: Omniwise [15] achieves 90% of predictions within 10% error on AMD MI250/MI300X.

Compiler cost models. TVM [8] and Ansor [49] use ML cost models (XGBoost, MLP) to guide autotuning, achieving ∼15% MAPE.

The TenSet dataset [50] (52M records) enables pre-trained models that accelerate autotuning 10×. TLP [47] uses deep learning for tensor program cost modeling. These tools prioritize ranking accuracy for schedule selection over absolute error.

5.3 Distributed Training and LLM Serving

Distributed systems introduce communication overhead, synchronization barriers, and parallelism strategy choices. Performance depends on the interplay between compute, memory, and network—requiring system-level modeling.

Training simulation. ASTRA-sim [42] provides end-to-end distributed training simulation using Chakra execution traces [38], with validated HGX-H100 configurations and pluggable network backends. It achieves 5–15% accuracy versus real clusters and enables exploration of parallelization strategies at scale. SimAI [?] provides full-stack LLM training simulation at Alibaba Cloud scale, achieving 98.1% alignment with production results. Lumos [29] targets LLM training through trace-driven modeling, achieving 3.3% error on H100 GPUs by capturing gradient accumulation, optimizer states, and activation checkpointing. Echo [6] simulates distributed training at scale. TrioSim [27] provides lightweight multi-GPU simulation. PRISM [?] uses probabilistic models for training performance prediction at 10K+ GPU scale.

Scaling and parallelism. The choice of parallelism strategy (data, tensor, pipeline, expert) critically impacts performance. Paleo [34] pioneered analytical estimation of training time by decomposing workloads into compute and communication components. MAD Max [16] extends analytical modeling to distributed systems. The Llama 3 scaling study [?] documents 4D parallelism at 16K H100 GPUs, providing ground truth for simulator validation. Sailor [?] addresses automated parallelism selection over heterogeneous clusters.

Inference serving. VIDUR [2] simulates LLM inference serving with scheduling strategies (vLLM [24], Orca [45], Sarathi [1]) without requiring GPU hardware. Frontier [14] extends to MoE and disaggregated inference with stage-centric simulation. ThrottLL'eM [18] models GPU throttling for energy-efficient LLM inference. These tools enable infrastructure planning and scheduling algorithm comparison at scale.

Memory system interactions. Memory increasingly dominates ML performance. KV cache management is the dominant LLM serving challenge; vLLM's PagedAttention [24] achieves 2–4× throughput improvement. VIDUR models cache allocation and eviction at the system level. Memory simulators (DRAMSim3 [28], Ramulator 2 [30]) provide detailed DRAM modeling but are primarily integrated with CPU/GPU simulators rather than used standalone for ML workloads.

5.4 Edge Device Modeling

Edge devices impose strict power, memory, and latency constraints. The diversity of edge hardware (mobile CPUs, GPUs, NPUs, DSPs) makes per-device analytical modeling impractical, leading to ML-augmented approaches.

nn-Meter [48] uses random forest ensembles with kernel-level feature engineering, reporting <1% MAPE. However, this claim is currently unverifiable: the tool's pre-trained predictors fail with modern scikit-learn versions due to pickle serialization changes, scoring only 3/10 in our reproducibility evaluation. LitePred [12] scales to 85 edge platforms using VAE-based intelligent sampling and transfer learning, achieving 0.7% MAPE with under one hour of adaptation per device. HELP [25] formulates cross-hardware prediction as meta-learning with MAML-style adaptation, achieving 1.9% MAPE with just 10 measurement samples on new devices. ESM [40] provides a framework for building effective surrogate models for hardware-aware neural architecture search.

The latency predictor study [11] provides the most systematic evaluation across approaches, showing transfer learning provides 22.5% average improvement, up to 87.6% on challenging cross-platform transfers.

5.5 Cross-Cutting Challenges

Several challenges cut across platform categories.

CNN-to-transformer gap. Nearly all reported accuracy numbers are measured on CNN workloads. Performance on transformers, MoE, and diffusion models is less well characterized. NeuSight is a notable exception, evaluating on GPT-3 inference, but most tools lack validated transformer support.

Kernel-to-end-to-end composition. Many tools predict kernel-level performance (nn-Meter, NeuSight), but composing kernel predictions into accurate end-to-end estimates is an unsolved problem. Memory allocation, kernel launch overhead, and inter-operator data movement introduce errors that compound across layers.

Static vs. profiling-based approaches. A fundamental practical divide exists between tools that predict from static specifications only (Timeloop, MAESTRO, Paleo) and those requiring runtime profiling data (Habitat, nn-Meter, HELP). Static approaches enable pre-silicon evaluation and NAS; profiling-based approaches achieve higher accuracy on existing hardware. This distinction is often more practically relevant than the analytical-vs-ML divide.

6 Comparison and Analysis

We analyze trade-offs across methodology types along four dimensions: accuracy, speed, generalization, and interpretability. Table 2 summarizes key characteristics.

6.1 Accuracy by Problem Difficulty

Rather than comparing accuracy numbers directly (which is misleading across different benchmarks, metrics, and hardware), we organize results by problem difficulty.

Accelerator dataflow modeling is the most amenable to accurate prediction because computations are regular and memory access patterns are statically determined. Timeloop achieves 5–10% error against RTL through purely analytical means.

Single-GPU kernel prediction for known architectures achieves 2–12% error through hybrid approaches (NeuSight, Habitat) that embed hardware-specific inductive biases.

Distributed system-level prediction achieves 2–15% error through trace-driven simulation (SimAI 1.9%, Lumos 3.3%, ASTRA-sim 5–15%), reflecting the challenge of modeling compute-communication interaction.

Table 2: Comparative analysis of representative tools across key dimensions. Accuracy figures are as reported in original papers; direct comparison is limited by differences in benchmarks, workloads, hardware targets, and evaluation protocols. †Unverifiable. *Surrogate fidelity, not hardware accuracy.

Tool	Methodology	Accuracy (reported)	Setup Cost	Generalization	Interpretability	Eval. Speed
Timeloop [32]	Analytical	5–10%	Arch spec only	Any accelerator	High	μs
MAESTRO [23]	Analytical	5–15%	Arch spec only	Any accelerator	High	μs
AMALI [7]	Analytical	23.6% MAPE	None	GPU LLM inference	High	ms
Accel-Sim [19]	Simulation	10–20%	GPU binary	GPU-specific	High	Hours
ASTRA-sim [42]	Trace-driven	5–15%	Execution trace	Configurable	Medium	Minutes
VIDUR [2]	Trace-driven	<5%	Kernel profiles	LLM-specific	High	Seconds
SimAI [?]	Trace-driven	1.9%	Full-stack setup	LLM training	Medium	Minutes
Lumos [29]	Trace-driven	3.3%	Execution trace	LLM training	Medium	Minutes
nn-Meter [48]	ML-augmented	<1%†	1K samples/kernel	Device-specific	Medium	ms
LitePred [12]	ML-augmented	0.7% MAPE	100 samples/device	85+ devices	Low	ms
HELP [25]	ML-augmented	1.9% MAPE	10 samples/device	Cross-platform	Low	ms
TVM [8]	ML-augmented	~15% MAPE	10K+	Operator-level	Medium	ms
NeuSight [26]	Hybrid	2.3% MAPE	Pre-trained	Cross-GPU	Medium	ms
Habitat [46]	Hybrid	11.8% MAPE	Online profiling	Cross-GPU	Medium	Per-kernel
ArchGym [22]	Hybrid	0.61% RMSE*	Simulation runs	Arch-specific	Medium	ms
Concorde [31]	Hybrid	2% CPI	Training corpus	Cross-μarch	Medium	ms

Cross-platform edge prediction achieves 0.7–2% error (LitePred, HELP) but requires per-device profiling data, trading generality for accuracy.

GPU analytical modeling remains the most difficult, with AMALI's 23.6% representing the current state of the art for purely analytical GPU LLM inference prediction—a problem where dynamic microarchitectural effects resist closed-form treatment.

6.2 Generalization Challenges

Workload generalization. Nearly all reported accuracy numbers are measured on CNN workloads. Cross-workload-type transfer (CNN→transformer) remains largely unvalidated. NeuSight is a notable exception, evaluating on LLM workloads, but most edge device predictors (nn-Meter, LitePred, HELP) are validated primarily on CNNs.

Hardware generalization. Three strategies show promise: meta-learning (HELP with 10-sample adaptation), feature-based transfer (LitePred across 85 devices), and analytical decomposition (Habitat separating compute/memory scaling). Cross-family transfer (GPU→TPU→PIM) remains unsolved.

Temporal generalization. Software stack evolution (framework updates, driver changes, compiler optimizations) invalidates trained models over time. No surveyed tool addresses continual learning for evolving software environments.

6.3 Interpretability and Design Insight

A key advantage of analytical models is actionable design insight. Timeloop identifies data movement bottlenecks; MAESTRO reveals suboptimal dataflow choices; VIDUR exposes scheduling inefficiencies. These insights directly guide design decisions.

ML-augmented approaches (nn-Meter, HELP) provide feature importance rankings but limited causal understanding. Hybrid approaches (NeuSight, Concorde) offer partial interpretability through their analytical components. The interpretability gap is practically significant: architects need to understand *why* a design is slow, not just predict *that* it is slow.

7 Open Challenges and Future Directions

7.1 Workload Coverage Gaps

Existing tools are primarily validated on CNN workloads. Transformers, mixture-of-experts (MoE), diffusion models, and dynamic inference patterns (e.g., AI agents with tool use [21]) remain underrepresented in validation benchmarks. LLM serving introduces variable sequence lengths (128–128K tokens) and dynamic batching that challenge static models. Scaling law prediction [? ?] connects model size to performance but does not address hardware-specific modeling.

7.2 The Composition Problem

Many tools predict kernel-level or operator-level performance, but composing these predictions into accurate end-to-end estimates is an unsolved problem. Memory allocation overhead, kernel launch latency, inter-operator data movement, and framework scheduling introduce compounding errors. For distributed systems, the composition extends across devices with communication overhead and synchronization. No surveyed tool provides validated composition guarantees.

7.3 Emerging Hardware Support

PIM architectures [? ? ? ?], neuromorphic processors, and analog compute present fundamentally different modeling challenges. Existing frameworks (Timeloop, MAESTRO) assume conventional memory hierarchies; PIM blurs the compute-memory boundary. Chiplet-based designs and disaggregated architectures introduce new interconnect modeling requirements.

7.4 Integration with Design Flows

Compiler integration (TVM, Ansor) needs uncertainty quantification for exploration-exploitation trade-offs. Architecture exploration (ArchGym) requires active learning for sample efficiency. LLM serving needs real-time prediction within microseconds; VIDUR provides offline simulation but online adaptation remains challenging. FlashAttention [10] and other hardware-aware algorithm optimizations change the performance landscape faster than models can be retrained.

7.5 Reproducibility and Trust

Our evaluation reveals a critical gap between reported accuracy and independently verifiable results. nn-Meter's claimed <1% MAPE is unverifiable because the tool cannot be run. Accuracy claims without reproducible evaluation are of limited value to practitioners. The community would benefit from standardized benchmarks with common workloads, hardware targets, and evaluation protocols.

7.6 Threats to Validity

Selection bias. Our literature search focused on top architecture venues (MICRO, ISCA, HPCA, ASPLOS) and systems venues (MLSys, OSDI, SOSP, NSDI), potentially under-representing work from application-specific venues, industry reports, or non-English publications.

Tool evaluation scope. Our reproducibility evaluation covers five tools (Timeloop, ASTRA-sim, VIDUR, nn-Meter, NeuSight), selected for coverage across methodology types and availability of open-source implementations. Results may not generalize to proprietary tools.

Metrics comparability. Accuracy figures use different metrics (MAPE, RMSE, Kendall's τ), benchmarks, and hardware targets. Tables 1 and 2 report metrics as stated in original papers; cross-paper comparisons should be interpreted with caution.

7.7 Future Directions

Five high-priority research opportunities: (1) **Transformer/MoE-aware tools**—current tools are validated on CNNs; attention and expert routing have distinct performance characteristics. (2) **Validated composition**—methods to compose kernel predictions into end-to-end estimates with bounded error. (3) **Unified energy-latency-memory prediction**—most tools focus on latency; edge and datacenter deployment need energy and memory modeling. (4) **Temporal robustness**—benchmarks for evaluating model accuracy under software stack evolution. (5) **Unified tooling**—no single tool addresses all needs; Docker-first deployment, portable model formats (ONNX), and composable modeling engines with standard workload representations (Chakra [38]) could reduce fragmentation.

8 Experimental Evaluation

We conducted hands-on reproducibility evaluations of five representative tools using a 10-point rubric: Setup (3 pts: Docker availability, clean installation, quick start), Reproducibility (4 pts: reference outputs, determinism, examples), and Usability (3 pts: API clarity, interpretability, maintenance). Table 3 summarizes results.

Table 3: Reproducibility evaluation scores (10-point rubric).

Tool	Setup	Reprod.	Usability	Total
Timeloop	3	4	2	9/10
ASTRA-sim	2.5	3	3	8.5/10
VIDUR	2.5	3.5	3	9/10
nn-Meter	2	0	1	3/10
NeuSight	2	3	2.5	7.5/10

Key findings. Docker-first tools (Timeloop, ASTRA-sim, VIDUR) scored 8.5+ by isolating dependencies; we executed all three on both x86_64 and aarch64 without issues. Timeloop provides reference outputs for all examples (Eyeriss, Simba) with deterministic results. ASTRA-sim includes validated HGX-H100 configurations; VIDUR enables scheduler comparison (vLLM, Orca, Sarathi) without GPU hardware. NeuSight's tile-based hybrid approach achieves 2.3% error on LLM workloads.

Critical anti-pattern. nn-Meter's pre-trained predictors fail with current scikit-learn due to pickle format changes—a cautionary example of ML model serialization fragility. Projects should prefer portable formats (ONNX) or pin exact dependency versions.

Best practices: (1) Provide Docker images to isolate dependencies; (2) Document Python version requirements; (3) Include reference outputs for validation; (4) Use portable model formats; (5) Pin dependency versions.

9 Conclusion

This survey analyzed over 50 tools and methods for modeling and predicting performance of ML workloads, spanning analytical models, cycle-accurate simulators, trace-driven simulation, and ML-augmented hybrid approaches.

Key findings: (1) Analytical frameworks (Timeloop, MAESTRO) remain the most effective for DNN accelerator design space exploration, offering microsecond evaluation with full interpretability. (2) Trace-driven simulators (ASTRA-sim, VIDUR, SimAI, Lumos) have emerged as the practical choice for system-level modeling of distributed training and LLM serving, achieving 2–15% accuracy at practical speeds. (3) Hybrid approaches combining analytical structure with learned components (NeuSight, Concorde) achieve the best accuracy on GPU kernel prediction (2–3% error). (4) LLM workloads require specialized modeling for prefill/decode phases, KV cache management, and multi-stage inference pipelines. (5) Reproducibility varies dramatically across tools—Docker-first tools score 8.5+ on our rubric while tools relying on serialized ML models risk becoming unusable.

Gaps and opportunities: Most tools are validated on CNN workloads; transformer/MoE validation is sparse. The kernel-to-end-to-end composition problem remains unsolved. Emerging hardware (PIM, chiplets) lacks mature modeling support. The community needs standardized benchmarks for cross-tool accuracy comparison.

As ML workloads grow in scale and diversity, accurate performance prediction becomes critical for efficient hardware design, system provisioning, and serving infrastructure planning. This survey provides practitioners a guide for selecting appropriate tools and researchers a roadmap for advancing the field.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramachandran. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 117–134.

[2] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramachandran. 2024. VIDUR: A Large-Scale Simulation Framework for LLM Inference. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–15.

[3] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 163–174. https://doi.org/10.1109/ISPASS.2009.4919648

[4] Abhimanyu Rajeshkumar Bambhaniya et al. 2025. HERMES: Understanding and Optimizing Multi-Stage AI Inference Pipelines. *arXiv preprint arXiv:2504.09775* (2025). Heterogeneous multi-stage LLM inference simulator with analytical modeling.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 Simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7. https://doi.org/10.1145/2024716.2024718

[6] Kai Cai, Wei Miao, Junyu Zhu, Jiaxu Chen, Hao Shan, Huanyu Li, and Chi Zhang. 2024. Echo: Simulating Distributed Training At Scale. *arXiv preprint arXiv:2412.12487* (2024).

[7] Zheng Cao et al. 2025. AMALI: An Analytical Model for Accurately Modeling LLM Inference on Modern GPUs. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1145/3695053.3731064 Reduces GPU LLM inference MAPE from 127.56% to 23.59% vs GCoM baseline.

[8] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 578–594.

[9] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. 16344–16359.

[11] Lukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2024. Latency Predictors for Neural Architecture Search. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–14.

[12] Yang Feng, Zhehao Li, Jiacheng Yang, and Yunxin Liu. 2024. LitePred: Transferable and Scalable Latency Prediction for Hardware-Aware Neural Architecture Search. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 1–18.

[13] Paraskevas Gavriilidis et al. 2025. LIFE: Forecasting LLM Inference Performance via Hardware-Agnostic Analytical Modeling. In *arXiv preprint arXiv:2508.00904*. Hardware-agnostic analytical model for LLM inference performance forecasting.

[14] Siddharth Ghosh et al. 2025. Frontier: Simulating the Next Generation of LLM Inference Systems. *arXiv preprint arXiv:2508.03148* (2025). Stage-centric simulator for MoE and disaggregated LLM inference, models expert parallelism and cross-cluster routing.

[15] Ameer Haj-Ali et al. 2025. Omniwise: Predicting GPU Kernels Performance with LLMs. *arXiv preprint arXiv:2506.20886* (2025). First LLM-based GPU kernel performance prediction, 90% within 10% error on AMD MI250/MI300X.

[16] Samuel Hsia, Kartik Chandra, and Kunle Olukotun. 2024. MAD Max Beyond Single-Node: Enabling Large Machine Learning Model Acceleration on Distributed Systems. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*. 753–766. https://doi.org/10.1109/ISCA59077.2024.00064

[17] Ryota Imai, Kentaro Harada, Ryo Sato, and Toshio Nakaike. 2024. Roofline-Driven Machine Learning for Large Language Model Performance Prediction. *NeurIPS Workshop on Machine Learning for Systems* (2024).

[18] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios Xydis, and Dimitrios Soudris. 2025. throttLL'eM: Predictive GPU Throttling for Energy Efficient LLM Inference Serving. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–14. Achieves up to 43.8% lower energy consumption for LLM inference.

[19] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*. 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[20] Jungho Kim et al. 2025. PyTorchSim: A Comprehensive, Fast, and Accurate NPU Simulation Framework. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14. https://doi.org/10.1145/3725843.3756045 PyTorch 2-integrated NPU simulator with custom RISC-V ISA and Tile-Level Simulation.

[21] Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. 2026. The Cost of Dynamic Reasoning: Demystifying AI Agents and Test-Time Scaling from an AI Infrastructure Perspective. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–14. HPCA 2026 (Jan 31–Feb 4, 2026, Las Vegas). First comprehensive system-level analysis of AI agents; quantifies resource usage, latency, and datacenter power consumption.

[22] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Norman P. Jouppi, Jignesh Parmar, Hyoukjun Kim, James Laudon, and Chandrakant Narayanaswami. 2023. ArchGym: An Open-Source Gymnasium for Machine Learning Assisted Architecture Design. In *Proceedings of the 50th International Symposium on Computer Architecture (ISCA)*. 1–16. https://doi.org/10.1145/3579371.3589049

[23] Hyoukjun Kwon, Prasanth Chatarasi, Michael Sarber, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2019. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. In *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14. https://doi.org/10.1145/3352460.3358292

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 611–626. https://doi.org/10.1145/3600006.3613165

[25] Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. 2021. HELP: Hardware-Adaptive Efficient Latency Prediction for NAS via Meta-Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34. 27016–27028.

[26] Seunghyun Lee, Amar Phanishayee, and Divya Mahajan. 2025. NeuSight: GPU Performance Forecasting via Tile-Based Execution Analysis. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1–15.

[27] Jianbo Li et al. 2025. TrioSim: A Lightweight Simulator for Large-Scale DNN Workloads on Multi-GPU Systems. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. 1–13. Multi-GPU DNN simulation with lightweight approach for distributed training analysis.

[28] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. In *IEEE Computer Architecture Letters*, Vol. 19. 106–109. https://doi.org/10.1109/LCA.2020.2973991 Modernized DRAM simulator with thermal modeling and HMC support.

[29] Wenxuan Liang et al. 2025. Lumos: Efficient Performance Modeling and Estimation for Large-scale LLM Training. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–16. Trace-driven performance modeling achieving 3.3% error on H100 GPUs for LLM training.

[30] Haocong Luo, Yahya Can Tugrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray Yağlıkcı, and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 22, 2 (2023), 129–132. https://doi.org/10.1109/LCA.2023.3333759 Modular DRAM simulator with DDR5, LPDDR5, HBM3, GDDR6 support and RowHammer mitigation modeling.

[31] Amir Nasr-Esfahany et al. 2025. Concorde: Fast and Accurate CPU Performance Modeling with Compositional Analytical-ML Fusion. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. 1–15. Hybrid analytical-ML approach achieving 2% CPI error at 5 orders of magnitude faster than gem5.

[32] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Muber, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[33] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aakanksha Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*. 118–132. https://doi.org/10.1109/ISCA59077.2024.00019 Best Paper Award.

[34] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance Model for Deep Neural Networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=SyVVJ85lg

[35] Saeed Rashidi, Srinivas Srinivasan, Kazem Hamedani, and Tushar Krishna. 2020. ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms. In *Proceedings of the IEEE International Symposium*

on Performance Analysis of Systems and Software (ISPASS). 81–92. https://doi.org/10.1109/ISPASS48437.2020.00018

[36] Alen Sabu, Harish Patil, Ameer Haj-Ali, and Trevor E. Carlson. 2022. LoopPoint: Checkpoint-driven Sampled Simulation for Multi-threaded Applications. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 606–618. https://doi.org/10.1109/HPCA53966.2022.00052 Extends sampling to multi-threaded applications with 2.3% error and up to 800x speedup.

[37] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 45–57. https://doi.org/10.1145/605397.605403 Introduces SimPoint: automatic selection of representative simulation points using k-means clustering.

[38] Srinivas Sridharan, Taekyung Heo, Jinwoo Choi, Garyfallia Yu, Saeed Rashidi, William Won, Zhaodong Meng, and Tushar Krishna. 2023. Chakra: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces. *arXiv preprint arXiv:2305.14516* (2023).

[39] Ondrej Sykora, Alexis Rucker, Charith Mendis, Rajkishore Barik, Phitchaya Mangpo Phothilimthana, and Saman Amarasinghe. 2022. GRANITE: A Graph Neural Network Model for Basic Block Throughput Estimation. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 1–13. https://doi.org/10.1109/IISWC55918.2022.00014

[40] various. 2025. ESM: A Framework for Building Effective Surrogate Models for Hardware-Aware Neural Architecture Search. In *Proceedings of the Design Automation Conference (DAC)*. 97.6% accuracy surrogate model framework for HW-aware NAS.

[41] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (2009), 65–76. https://doi.org/10.1145/1498765.1498785

[42] William Won, Taekyung Heo, Saeed Rashidi, Saeed Talati, Srinivas Srinivasan, and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-Model Training at Scale. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 283–294. https://doi.org/10.1109/ISPASS57527.2023.00035

[43] Yannan Nellie Wu, Joel Emer, and Vivienne Sze. 2022. Sparseloop: An Analytical Approach to Sparse Tensor Accelerator Modeling. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–15. https://doi.org/10.1109/MICRO56248.2022.00078

[44] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. 2003. SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)*. 84–97. https://doi.org/10.1109/ISCA.2003.1206991 Statistical sampling achieving 0.64% CPI error with 35x speedup over detailed simulation.

[45] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. ORCA: A Distributed Serving System for Transformer-Based Generative Models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 521–538.

[46] Geoffrey X. Yu, Yubo Gao, Pavel Golber, and Asaf Cidon. 2021. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 503–521.

[47] Yi Zhai, Yu Cheng Wang, Peng Jiang, and Congming Kang. 2023. TLP: A Deep Learning-based Cost Model for Tensor Program Tuning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 833–845. https://doi.org/10.1145/3575693.3575736

[48] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 81–93. https://doi.org/10.1145/3458864.3467882 Best Paper Award.

[49] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 863–879.

[50] Lianmin Zheng, Ruochen Liu, Junru Shao, Tianqi Chen, Joseph E. Gonzalez, Ion Stoica, and Zhihao Zhang. 2021. TenSet: A Large-scale Program Performance Dataset for Learned Tensor Compilers. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34. 29876–29888.

[51] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianyu Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 1–18.