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

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

Anonymous Author(s)
Under Review
Anonymous

Abstract

As machine learning workloads grow in scale and complexity—spanning training and inference for CNNs, transformers, mixture-of-experts models, and LLMs—architects and system designers need fast, accurate methods to predict their performance across diverse hardware platforms. This survey provides a comprehensive analysis of the tools and methods available for modeling and simulating the performance of ML workloads, covering analytical models, cycle-accurate simulators, trace-driven approaches, and ML-augmented hybrid techniques. We survey over 30 tools drawn from 53 papers across architecture venues (MICRO, ISCA, HPCA, ASPLOS) and systems venues (MLSys, OSDI, NSDI) 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 three dimensions—methodology type (analytical, simulation, ML-augmented, hybrid), target platform (accelerators, GPUs, distributed systems, edge devices), and abstraction level (kernel, model, system)—while additionally characterizing tools by workload coverage, revealing a pervasive CNN-validation bias. 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 of five representative tools, finding that reproducibility varies dramatically: Docker-first tools score 8.5+/10 on our rubric while tools relying on serialized ML models risk becoming unusable. 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 the field of ML workload performance prediction.

Keywords

ML workload performance prediction, DNN accelerator modeling, GPU simulation, distributed training simulation, LLM inference serving, 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. The shift toward domain-specific architectures [22], from Google's TPU [30, 31] to custom training accelerators, has created a heterogeneous hardware landscape where 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 this increasingly heterogeneous landscape of GPUs, TPUs, custom accelerators, and multi-device clusters.

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 [50] and MAESTRO [38] model DNN accelerator performance through closed-form data movement analysis, achieving 5–10% error versus RTL at microsecond evaluation speed. Cycle-accurate simulators like GPGPU-Sim [4] and Accel-Sim [34] provide detailed GPU modeling but require hours per workload. Trace-driven simulators like ASTRA-sim [71] and VIDUR [3] target distributed training and LLM serving at system scale. ML-augmented approaches like NeuSight [42] 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—the architect or engineer who needs to select a modeling tool for a specific design or deployment task. Existing surveys focus on ML *techniques* for performance modeling [65] or on specific hardware targets [50], leaving practitioners without guidance on which tools suit their needs across the full modeling spectrum. This survey fills that gap by providing a methodology-centric view of the tools and methods available for predicting ML workload performance.

We make the following contributions:

- A **methodology-centric taxonomy** organizing tools along three dimensions: methodology type (analytical, simulation, ML-augmented, hybrid), target platform (DNN accelerators, GPUs, distributed systems, edge devices), and abstraction level (kernel, model, system), with a quantitative coverage matrix identifying research gaps and a workload coverage analysis exposing the CNN-validation bias in the literature.
- A **systematic survey** of over 30 modeling tools drawn from 53 papers across architecture venues (MICRO, ISCA,

MICRO 2026, Austin, TX, USA
2026.

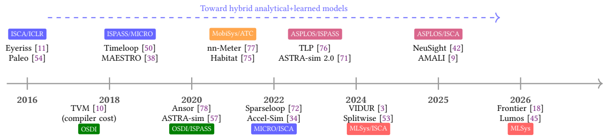

Figure 1: Evolution of performance modeling tools for ML workloads (2016–2026). Early analytical frameworks (Eyeriss, Paleo) gave way to systematic accelerator modeling (Timeloop, MAESTRO) and distributed training simulation (ASTRA-sim). ML-augmented approaches (TVM, Habitat, NeuSight) learn performance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

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 careful qualification of paper-reported accuracy claims and identification of cases where reported numbers are unverifiable.
- **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 and positions this work relative to existing surveys. 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 and a practitioner tool selection guide. Section 7 presents hands-on reproducibility evaluations. Section 8 discusses open challenges and future directions. Section 9 concludes.

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.

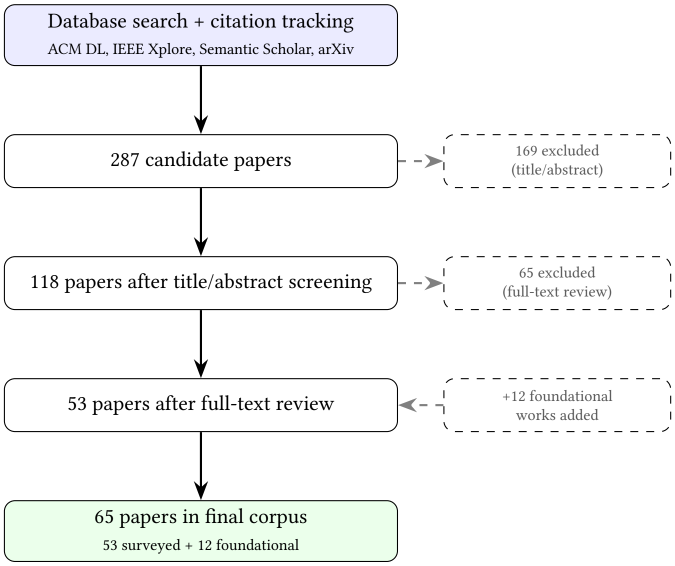
Database search + citation tracking
ACM DL, IEEE Xplore, Semantic Scholar, arXiv
287 candidate papers
169 excluded
(title/abstract)
118 papers after title/abstract screening
65 excluded
(full-text review)
53 papers after full-text review
+12 foundational
works added
65 papers in final corpus
53 surveyed + 12 foundational

Figure 2: Paper selection process. Initial database searches and citation tracking yielded 287 candidates; title/abstract screening and full-text review reduced the set to 53 papers meeting all inclusion criteria. 12 foundational works were added for context, yielding 65 papers in the final corpus.

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. Figure 2 summarizes the selection process.

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

Classification. We classify each paper along three dimensions: *methodology type* (analytical, cycle-accurate simulation, trace-driven simulation, ML-augmented, or hybrid), *target platform* (DNN accelerator, GPU, distributed system, edge device, or CPU), and *abstraction level* (kernel/operator, model/end-to-end, or system). We additionally characterize each tool by workload coverage, prediction targets, and reported accuracy metrics.

2.1 Related Surveys

Several surveys address adjacent topics. In the ML-for-systems space, Rakhshanfar and Zarandi [56] survey ML techniques for processor design space exploration, focusing on surrogate model construction rather than the tools available to ML practitioners. Sze et al. [66] provide a comprehensive treatment of DNN hardware architectures and dataflow optimization, establishing the conceptual framework on which analytical tools like Timeloop and MAESTRO are built; however, their scope is DNN accelerator design rather than cross-platform performance prediction. In GPU simulation, the gem5-gpu [6] and GPGPU-Sim [4] ecosystems have generated extensive evaluation literature, but no survey organizes GPU, accelerator, distributed, and edge modeling tools within a unified taxonomy. The MLPerf benchmark suites [47, 59] standardize ML workload measurement across hardware but focus on *measurement* rather than *prediction*—they provide ground truth data that performance models should target but do not survey the modeling tools themselves. Hennessy and Patterson [22] frame the current era as a "new golden age" for domain-specific architectures, motivating the need for performance prediction tools that span the heterogeneous hardware landscape, but do not survey these tools.

This survey differs from prior work in three ways: (1) it spans the full methodology spectrum from analytical to ML-augmented, rather than focusing on a single approach; (2) it covers all major target platforms (accelerators, GPUs, distributed systems, edge devices) rather than a single hardware class; and (3) it includes hands-on reproducibility evaluations that go beyond paper-reported accuracy claims. The closest prior work is the latency predictor study by Dudziak et al. [15], which systematically compares edge device predictors for NAS; we broaden the scope to the full platform and methodology landscape.

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. Modern ML frameworks like PyTorch [52] and TensorFlow [1] define workloads as computation graphs of operators, providing a structured representation that performance models can exploit.

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 [39].

Scale and distribution. Large model training distributes computation across thousands of GPUs using data, tensor, pipeline, and expert parallelism [13]. 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) [53]. Effective modeling must capture both phases and their interaction under batched serving [2, 74].

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 [70] 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 [50] 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 [6] (CPUs), GPGPU-Sim [4] (GPUs), and Accel-Sim [34] (modern GPUs) achieve detailed accuracy but suffer 1000–10000× slowdown, making them impractical for full ML workload evaluation. Sampling techniques (SimPoint [61], SMARTS [73]) 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 [71] models distributed training using Chakra execution traces [63] with pluggable compute, memory, and network backends. VIDUR [3] 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 [77], XGBoost in TVM [10]) to deep learning (NeuSight [42]) and meta-learning (HELP [41]). 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.

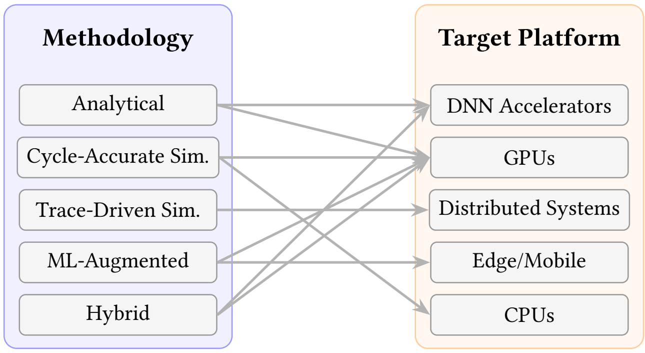

Figure 3: Primary dimensions of the taxonomy for ML workload performance modeling. The primary axis is methodology type (how performance is predicted); the secondary axis is target platform. The third dimension, abstraction level, is shown separately in Figure 4. Arrows show dominant pairings: analytical models for accelerators, cycle-accurate simulation for GPUs/CPUs, trace-driven simulation for distributed systems, and ML-augmented approaches for edge devices and compiler cost models.

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 three dimensions. The *primary axis* is methodology type—how a tool predicts performance—because methodology determines the fundamental trade-offs between accuracy, speed, interpretability, and data requirements. The *secondary axes* are target platform and abstraction level, which together determine the scope and applicability of each tool. We additionally characterize tools by workload coverage, exposing a pervasive CNN-validation bias in the literature.

Figure 3 illustrates the primary and secondary dimensions. Table 1 provides a unified view combining the coverage matrix (number of surveyed tools per methodology–platform cell) with trade-off profiles (evaluation speed, data requirements, interpretability, and failure modes), with empty cells highlighting research gaps.

Table 1 reveals three structural observations. First, trace-driven simulation is exclusively used for distributed systems—no surveyed tool applies trace-driven methods to single-device GPU or accelerator modeling, despite the potential for trace-driven approaches to avoid the slowdown of cycle-accurate simulation while retaining more fidelity than analytical models. Second, edge/mobile devices are served exclusively by ML-augmented approaches; the absence of analytical or hybrid models for edge devices reflects the hardware diversity problem but also represents a research gap, since hybrid approaches could combine the interpretability of analytical models with the adaptability of learned components. Third, no ML-augmented or hybrid tool specifically targets distributed system modeling—tools like VIDUR use ML internally for kernel prediction but are architecturally trace-driven simulators. The trade-off columns further show that methodologies cluster into two speed regimes: sub-millisecond (analytical, ML-augmented, hybrid) suitable for design space exploration, and minutes-to-hours (simulation, trace-driven) suitable for detailed validation.

4.1 Primary Axis: Methodology Type

The choice of methodology determines fundamental trade-offs between accuracy, evaluation speed, data requirements, and interpretability, as summarized in the right columns of Table 1.

4.1.1 Analytical Models. Analytical models express performance as closed-form functions of workload and hardware parameters. For DNN accelerators, Timeloop [50] models data movement across memory hierarchies for any valid loop-nest mapping, achieving 5–10% error versus RTL at 2000× speedup. MAESTRO [38] provides data-centric dataflow analysis using intuitive directives. Sparseloop [72] extends to sparse tensor operations. Paleo [54] pioneered layer-wise analytical modeling for DNNs, decomposing networks into compute and communication components for distributed training prediction. AMALI [9] 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 accuracy ceiling of analytical approaches for complex GPU workloads—the residual error stems from dynamic microarchitectural effects that resist closed-form treatment (see §5.2 for detailed analysis).

4.1.2 Cycle-Accurate Simulation. Cycle-accurate simulators model hardware at register-transfer level, providing the highest fidelity. gem5 [6] (CPUs), GPGPU-Sim [4] (GPUs), and Accel-Sim [34] (modern NVIDIA GPUs, SASS-level trace-driven) achieve 0.90–0.97 IPC correlation. PyTorchSim [35] 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 [61], SMARTS [73], LoopPoint [60]) accelerate general-purpose workload simulation but are not specifically validated for ML workload patterns. Recent work on dissecting modern GPU cores [27] has improved Accel-Sim's accuracy to 13.98% MAPE by reverse-engineering undocumented microarchitectural details. Note that the 0.90–0.97 IPC *correlation* metric can coexist with 20%+ absolute latency error for workloads with atypical occupancy patterns—correlation captures relative ordering fidelity but not absolute prediction accuracy.

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 [71] models distributed training end-to-end using Chakra execution traces [63], with pluggable compute, memory, and network backends, achieving 5–15% error versus real clusters. Echo [7] simulates distributed training at scale using analytical compute models with network

Table 1: Methodology taxonomy: coverage matrix and trade-off profile. Platform columns show the number of surveyed tools per cell; 0 indicates an explicit research gap. Speed, data requirements, and interpretability determine practical applicability; the failure mode column identifies the primary condition under which each methodology breaks down.

Methodology	DNN Accel.	GPU	Distrib. Systems	Edge/ Mobile	CPU	Eval. Speed	Data Req.	Interp.	Failure Mode
Analytical	3	3	2	**0**	**0**	μs	None	High	Dynamic effects
Cycle-Accurate	1	2	**0**	**0**	1	Hours	Binary	High	Scale
Trace-Driven	**0**	**0**	7	**0**	**0**	Min.	Traces	Med.	Trace fidelity
ML-Augmented	**0**	3	**0**	3	1	ms	Profiling	Low	Distrib. shift
Hybrid	1	2	**0**	**0**	1	ms	Mixed	Med.	Training domain

simulation. Lumos [45] targets LLM training performance through trace-driven modeling, achieving 3.3% error on H100 GPUs.

For LLM inference serving, VIDUR [3] provides discrete-event simulation capturing prefill/decode phases, KV cache management, and request scheduling (Orca [74], Sarathi [2] strategies) with <5% error. Frontier [18] extends to MoE and disaggregated inference with stage-centric simulation. SimAI [68] 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—we classify by architectural design intent rather than implementation detail.

4.1.4 ML-Augmented Models. ML-augmented approaches learn performance functions entirely from profiling data, without embedding analytical domain knowledge. nn-Meter [77] uses random forest ensembles with kernel-level feature engineering for edge device latency prediction. LitePred [16] scales to 85 edge platforms using VAE-based intelligent sampling and transfer learning. HELP [41] formulates cross-hardware prediction as meta-learning, achieving adaptation with just 10 samples on new devices. TVM [10] and Ansor [78] use XGBoost/MLP cost models to guide compiler autotuning, with the TenSet dataset [79] (52M records) enabling pre-trained models.

ML-augmented approaches excel when sufficient profiling data is available and the training distribution matches deployment conditions. Their critical failure mode is *silent distribution shift*: a model trained on CNN kernels may produce confident but wrong predictions for transformer attention kernels, with no built-in mechanism to flag out-of-distribution inputs. 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 and how unverifiable accuracy claims should be discounted by practitioners.

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 [42] uses tile-based prediction mirroring CUDA's execution model, achieving 2.3% error on GPT-3 inference. Concorde [49] fuses analytical models with learned corrections for CPU performance at 2% CPI error. Habitat [75] decomposes execution into analytically-modeled compute and memory components. ArchGym [37] connects ML optimization to analytical simulators for design space exploration.

The latency predictor study [15] demonstrates that hybrid approaches with transfer learning achieve 22.5% average improvement over baselines. Note that cross-tool accuracy comparisons require careful contextualization—we discuss methodological caveats (surrogate fidelity vs. real hardware error, evaluation-era fairness) in §5.2 and §5.5.

4.2 Secondary Axis: Target Platform

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

DNN Accelerators (systolic arrays, dataflow architectures), from Google's TPU [30, 31] to custom ASICs, 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 [29, 70]) 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 [65] target CPU performance but focus on general-purpose workloads rather than ML-specific patterns.

4.3 Secondary Axis: Abstraction Level

The abstraction level at which a tool operates determines what it can predict and where composition errors arise.

Kernel/Operator-level tools predict the latency of individual kernels or DNN operators (NeuSight, nn-Meter, TVM, GRANITE).

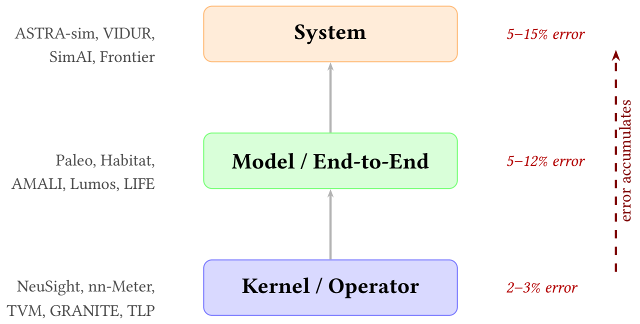
ASTRA-sim, VIDUR, SimAI, Frontier
System
5–15% error
Paleo, Habitat, AMALI, Lumos, LIFE
Model / End-to-End
5–12% error
NeuSight, nn-Meter, TVM, GRANITE, TLP
Kernel / Operator
2–3% error
error accumulates

Figure 4: Abstraction level hierarchy and the composition problem. Tools operate at one of three levels; composing predictions across levels accumulates error. Error ranges are representative values from surveyed papers.

They achieve the highest accuracy because the prediction scope is narrowly defined, but composing kernel predictions into end-to-end model latency introduces errors from memory allocation, kernel launch overhead, and inter-operator data movement.

Model/End-to-End tools predict full model inference or training time (Paleo, Habitat, AMALI, Lumos). They must account for graph-level effects (operator fusion, memory planning, scheduling) that kernel-level tools ignore, typically at the cost of higher error.

System-level tools predict multi-device or serving system performance (ASTRA-sim, VIDUR, SimAI, Frontier). They capture communication, scheduling, and resource contention effects but depend on the accuracy of their compute sub-models—creating a composition chain where kernel-level errors propagate through model-level to system-level predictions.

This three-level hierarchy makes explicit the *composition problem*: most tools operate at one level, but practitioners need predictions that span levels. The gap between kernel-level error (2–3% for NeuSight) and system-level error (5–15% for ASTRA-sim) reflects both the inherent difficulty of system modeling and the error accumulated through composition. Figure 4 illustrates this hierarchy and the error ranges at each level.

4.4 Workload Coverage

Table 2 characterizes the workload types on which each tool has been validated, exposing a pervasive CNN-validation bias.

The workload coverage table reveals that **no surveyed tool has been validated on diffusion models or dynamic inference workloads** (e.g., AI agents with tool use [36]). Only Frontier [18] has validated MoE support. For transformers, NeuSight, AMALI, VIDUR, and Frontier provide validated coverage, but each targets a different platform and abstraction level—no single tool offers validated transformer performance prediction across the full stack from kernel to system level. This workload coverage gap is the most actionable finding of our taxonomy: practitioners working with non-CNN workloads must either (a) accept unvalidated predictions from CNN-trained tools, (b) collect their own validation data, or (c) fall back to measurement.

Table 2: Workload validation coverage. ✓ = validated in the original paper; ○ = partial or indirect validation; — = no validation. Nearly all tools report accuracy on CNN workloads; transformer and MoE coverage is sparse. Empty columns (diffusion, dynamic inference) represent workload types with *no* validated performance modeling tools.

Tool	CNN	Trans-former	LLM Train	MoE	Diff.
Timeloop	✓	○	—	—	—
MAESTRO	✓	—	—	—	—
NeuSight	✓	✓	—	—	—
Habitat	✓	—	—	—	—
AMALI	—	✓	—	—	—
ASTRA-sim	✓	○	✓	—	—
VIDUR	—	✓	—	—	—
SimAI	—	—	✓	—	—
Lumos	—	—	✓	—	—
Frontier	—	✓	—	✓	—
nn-Meter	✓	—	—	—	—
LitePred	✓	—	—	—	—
HELP	✓	—	—	—	—
TVM/Ansor	✓	○	—	—	—

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 3 provides a comprehensive comparison.

5.1 DNN Accelerator Modeling

DNN accelerators employ specialized dataflows and memory hierarchies optimized for tensor operations [66]. The regularity of DNN computations makes this domain particularly amenable to analytical modeling.

Analytical frameworks dominate accelerator modeling. Timeloop [50] analytically computes data reuse, latency, and energy from loop-nest representations, achieving 5–10% error versus RTL simulation at 2000× speedup. It provides reference outputs for standard accelerator designs (Eyeriss [11], Simba) with deterministic results—a key reproducibility strength. MAESTRO [38] offers data-centric dataflow directives that simplify specification but is less precise than Timeloop for detailed energy modeling. Sparseloop [72] extends Timeloop to sparse tensor operations by modeling the interaction between sparsity patterns (structured vs. unstructured), compression formats (CSR, bitmap), and hardware decompression/intersection units. This is critical for efficient transformer inference where attention matrices exhibit structured sparsity, but Sparseloop assumes static, known sparsity distributions—dynamic sparsity patterns from techniques like token pruning or dynamic routing in MoE models fall outside its modeling capability.

Simulation approaches. PyTorchSim [35] integrates PyTorch 2 with cycle-accurate NPU simulation, supporting custom RISC-V ISA and systolic arrays with configurable memory hierarchies. Unlike standalone accelerator simulators, PyTorchSim directly consumes

Table 3: 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 [50]	NPU	A	Latency/Energy	5–10%	μs	Loop-nest DSE
MAESTRO [38]	NPU	A	Latency/Energy	5–15%	μs	Data-centric directives
Sparseloop [72]	NPU	A	Sparse tensors	5–10%	μs	Compression modeling
PyTorchSim [35]	NPU	S	Cycle-accurate	N/A[‡]	Hours	PyTorch 2 integration
ArchGym [37]	Multi	H	Multi-objective	0.61%[*]	ms	ML-aided DSE
GPU Performance Modeling						
Accel-Sim [34]	GPU	S	Cycle-accurate	10–20%	Hours	SASS trace-driven
GPGPU-Sim [4]	GPU	S	Cycle-accurate	10–20%	Hours	CUDA workloads
AMALI [9]	GPU	A	LLM inference	23.6%	ms	Memory hierarchy
NeuSight [42]	GPU	H	Kernel/E2E latency	2.3%	ms	Tile-based prediction
Habitat [75]	GPU	H	Training time	11.8%	Per-kernel	Wave scaling
Distributed Training and LLM Serving						
ASTRA-sim [71]	Distributed	T	Training time	5–15%	Minutes	Collective modeling
SimAI [68]	Distributed	T	Training time	1.9%	Minutes	Full-stack simulation
Lumos [45]	Distributed	T	LLM training	3.3%	Minutes	H100 training
VIDUR [3]	GPU cluster	T	LLM serving	<5%	Seconds	Prefill/decode phases
Frontier [18]	Distributed	T	MoE inference	—	Minutes	Stage-centric sim.
TrioSim [43]	Multi-GPU	T	DNN training	N/A[‡]	Minutes	Lightweight multi-GPU
Edge Device Modeling						
nn-Meter [77]	Edge	M	Latency	<1%[†]	ms	Kernel detection
LitePred [16]	Edge	M	Latency	0.7%	ms	85-platform transfer
HELP [41]	Multi	M	Latency	1.9%	ms	10-sample adaptation
Compiler Cost Models						
TVM [10]	GPU	M	Schedule perf.	~15%	ms	Autotuning guidance
Ansor [78]	GPU	M	Schedule perf.	~15%	ms	Program sampling
TLP [76]	GPU	M	Tensor program	<10%	ms	Transformer cost model

PyTorch computation graphs, eliminating the manual workload translation step that introduces errors and limits adoption. However, it does not report accuracy against real hardware, and its cycle-accurate approach inherits the speed limitations of simulation-based methods, making it impractical for large-model evaluation.

ML-augmented design. ArchGym [37] 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, as they blur the compute-memory boundary that conventional frameworks assume. Early PIM modeling tools [23, 28, 40, 51] target attention acceleration and heterogeneous PIM-GPU co-simulation, but none report accuracy against real PIM hardware—we discuss PIM modeling gaps further in Section 8.3.

Synthesis. Accelerator modeling is the most mature subdomain surveyed, with Timeloop's analytical framework achieving a favorable balance of accuracy (5–10% error), speed, and interpretability that has made it the de facto standard for accelerator design space exploration. The progression from Timeloop through Sparseloop to PIM-aware tools illustrates a recurring pattern: each extension addresses a new workload characteristic (sparsity, near-memory compute) but adds modeling complexity that erodes the simplicity advantage of analytical approaches. The key gap is cycle-accurate validation—ArchGym and PyTorchSim provide simulation-based alternatives, but neither validates against manufactured silicon, leaving the accuracy of all accelerator modeling tools ultimately anchored to RTL comparisons rather than measured hardware.

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 [4] and Accel-Sim [34] achieve 0.90–0.97 IPC correlation through detailed microarchitectural modeling. Recent work reverse-engineering modern GPU cores [27] 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 [70] provides a useful upper bound but misses occupancy and memory hierarchy effects. Roofline-LLM [29] extends roofline analysis to LLM inference. AMALI [9] 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 [42] 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 [75] decomposes execution into analytically-modeled compute and memory components using wave scaling analysis, achieving 11.8% error for cross-GPU transfer (e.g., V100→A100). Its key insight is that compute and memory bandwidth scale independently across GPU generations, enabling prediction on new hardware from profiling data on existing hardware. However, Habitat requires source GPU profiling, limiting its use for pre-silicon design exploration, and its wave scaling model assumes that GPU occupancy patterns remain similar across generations—an assumption that breaks for workloads with qualitatively different memory access patterns (e.g., KV cache in LLM decode vs. activation reuse in CNN training). 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 [53, 80]. VIDUR [3] provides discrete-event simulation for LLM serving systems, capturing request scheduling strategies (Orca [74], Sarathi [2]) with <5% error. LIFE [17] offers hardware-agnostic analytical LLM inference modeling by decomposing inference into compute and memory-access components that can be parameterized for arbitrary hardware specifications, enabling performance prediction without hardware-specific profiling. Its hardware-agnostic design enables pre-silicon evaluation of new accelerators for LLM workloads, but the analytical approach shares the same accuracy limitations as AMALI when applied to GPUs with complex dynamic behavior. HERMES [5] targets heterogeneous multi-stage LLM inference pipelines where different model components (embedding, attention, FFN) execute on different hardware, modeling the inter-stage data transfer and load balancing that arise in disaggregated serving architectures. Emerging work uses LLMs for GPU kernel performance prediction: Omniwise [21] achieves 90% of predictions within 10% error on AMD MI250/MI300X, and SwizzlePerf [67] achieves 2.06× speedup through hardware-aware spatial optimization.

Compiler cost models. TVM [10] and Ansor [78] use ML cost models (XGBoost, MLP) to guide autotuning, achieving ~15% MAPE. The TenSet dataset [79] (52M records) enables pre-trained models that accelerate autotuning 10×. TLP [76] uses deep learning (transformer-based architecture) for tensor program cost modeling, specifically designed for the irregular computation patterns in transformer workloads that challenge traditional XGBoost cost models; it achieves <10% MAPE on transformer operators where TVM's default cost model shows higher error, demonstrating that workload-specific cost models can significantly improve autotuning for non-CNN workloads. SynPerf [69] takes a complementary approach, using performance models to guide GPU kernel synthesis rather than merely evaluating existing kernels. Note that SwizzlePerf and SynPerf are primarily *consumers* of performance models—they use performance predictions to guide optimization and synthesis, rather than providing general-purpose performance prediction themselves. These tools prioritize ranking accuracy for schedule selection over absolute error.

Synthesis. GPU modeling exhibits the widest methodological spread of any platform category: cycle-accurate simulation (Accel-Sim), analytical models (AMALI, roofline), hybrid learned approaches (NeuSight, Habitat), and LLM-based predictors (Omniwise) all target the same hardware with error rates spanning 2%–24%. This diversity reflects the fundamental tension in GPU modeling: the microarchitectural complexity that makes GPUs powerful also makes them hard to model analytically, while the rapid hardware evolution that motivates prediction also invalidates training data for learned approaches. NeuSight's tile-based decomposition currently offers the best accuracy–speed trade-off for LLM workloads, but its reliance on per-GPU profiling limits pre-silicon use—a gap that analytical approaches like AMALI and LIFE fill despite higher error. The compiler cost model ecosystem (TVM, TLP, SynPerf) represents a distinct use case where relative ranking matters more than absolute prediction, suggesting that evaluation criteria should be workload-aware.

5.3 Distributed Training and LLM Serving

Distributed systems introduce communication overhead, synchronization barriers, and parallelism strategy choices. Modern large model training uses multiple parallelism dimensions: tensor parallelism splits individual layers across GPUs [62], pipeline parallelism distributes layers across pipeline stages [26], and memory-efficient optimizers like ZeRO [55] partition optimizer states across data-parallel workers. Performance depends on the interplay between compute, memory, and network—requiring system-level modeling.

Training simulation. ASTRA-sim [71] provides end-to-end distributed training simulation using Chakra execution traces [63], with validated HGX-H100 configurations and pluggable network backends. It achieves 5–15% error versus real clusters and enables exploration of parallelization strategies at scale. SimAI [68] provides full-stack LLM training simulation at Alibaba Cloud scale, modeling compute, memory, network, and collective communication in an integrated framework that achieves 1.9% MAPE versus production training runs. Its key differentiator is validation against production training runs at datacenter scale—most simulators validate at 4–64 GPU configurations, whereas SimAI validates against thousands of GPUs where network congestion and load imbalance effects dominate. Lumos [45] targets LLM training through trace-driven modeling, achieving 3.3% error on H100 GPUs by capturing gradient accumulation, optimizer states, and activation checkpointing. Echo [7] combines analytical compute models with packet-level network simulation for collective communication evaluation, though it does not report end-to-end accuracy against real hardware. TrioSim [43] offers lightweight multi-GPU simulation through selective fidelity, enabling rapid what-if analysis for multi-GPU configurations but without real-hardware accuracy baselines. PRISM [19] produces prediction intervals rather than point estimates at 10K+ GPU scale, capturing the stochastic performance variation (network congestion, stragglers) that deterministic simulators miss.

Scaling and parallelism. The choice of parallelism strategy (data, tensor, pipeline, expert) critically impacts performance. Paleo [54] pioneered analytical estimation of training time by decomposing workloads into compute and communication components. MAD Max [25] decomposes training time into compute, communication, and memory components per parallelism dimension, enabling rapid analytical evaluation of parallelism configurations without simulation. The Llama 3 scaling study [13] documents 4D parallelism at 16K H100 GPUs, providing ground truth for simulator validation. Sailor [64] addresses automated parallelism selection over heterogeneous clusters, where GPUs of different generations or types must be jointly scheduled—a problem that most simulators cannot model because they assume homogeneous hardware.

Inference serving. VIDUR [3] simulates LLM inference serving with scheduling strategies (vLLM [39], Orca [74], Sarathi [2]) without requiring GPU hardware. Frontier [18] extends to MoE and disaggregated inference with stage-centric simulation. ThrottLL'eM [32] models the interaction between GPU power management (frequency throttling, power capping) and LLM inference performance, addressing a dimension that most tools ignore: real GPU deployments operate under power and thermal constraints that reduce effective performance below theoretical peaks. By modeling throttling effects, ThrottLL'eM enables energy-efficient inference scheduling that trades latency headroom for power savings—a critical concern for datacenter operators where energy costs dominate TCO. Recent LLM inference optimizations also change the performance characteristics that simulators must capture: for example, MEDUSA [8] introduces speculative decoding that transforms sequential token generation into parallel verification, fundamentally altering the compute-to-memory ratio that models like VIDUR assume. Such optimizations illustrate a moving-target challenge: performance models must track not just hardware evolution but algorithmic innovations that restructure execution patterns. These tools collectively enable infrastructure planning and scheduling algorithm comparison at scale.

Memory system interactions. Memory increasingly dominates ML performance: KV cache management is the key LLM serving bottleneck (vLLM's PagedAttention [39] achieves 2–4× throughput improvement), and VIDUR models cache allocation and eviction at the system level. Low-level memory simulators (DRAMSim3 [44], Ramulator 2 [46]) integrate with tools like Accel-Sim rather than being used standalone for ML workloads.

Synthesis. Distributed system modeling is the fastest-growing subdomain, with six new tools published since 2024 (SimAI, Lumos, Echo, TrioSim, PRISM, Sailor). This surge reflects the practical urgency: training runs on thousands of GPUs cost millions of dollars, making pre-deployment performance prediction economically critical. The tools bifurcate into two design philosophies: *trace-driven fidelity* (ASTRA-sim, SimAI) replays recorded execution traces through pluggable backends for maximum realism, while *analytical decomposition* (Paleo, MAD Max) trades fidelity for speed and interpretability. PRISM's probabilistic approach represents an emerging third path, acknowledging that large-scale systems are inherently stochastic. The inference serving tools (VIDUR, Frontier, ThrottLL'eM) face a distinct challenge: algorithmic innovations like speculative decoding [8] continuously alter the performance characteristics that models must capture, creating a moving-target problem absent in training simulation.

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 [77] 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 [16] 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. Its key innovation is intelligent sample selection: rather than profiling all operators on a new device, LitePred uses a VAE to identify the most informative operators to profile, reducing adaptation cost by an order of magnitude. However, the "85 platforms" are predominantly ARM-based mobile CPUs and GPUs—the diversity within this evaluation set is unclear, and transfer to fundamentally different accelerators (NPUs, DSPs) likely degrades significantly. HELP [41] formulates cross-hardware prediction as meta-learning with MAML-style adaptation, achieving 1.9% MAPE with just 10 measurement samples on new devices. The 10-sample adaptation is compelling for rapid deployment but raises a selection problem: which 10 operators to profile depends on the target workload, and suboptimal sample selection can significantly degrade accuracy on workload-critical operators not represented in the adaptation set. ESM [48] provides a systematic framework for building effective surrogate models for hardware-aware neural architecture search (NAS), evaluating multiple ML architectures (MLPs, gradient-boosted trees, GNNs) as latency surrogates across different hardware platforms. Its key finding is that model architecture choice matters less than training data quality and feature engineering—well-tuned random forests match or outperform deep learning surrogates, challenging the assumption that more complex models yield better hardware-aware NAS performance. This result has practical implications for the ML-augmented tools surveyed here: the accuracy gains from sophisticated model architectures may be marginal compared to improvements in profiling data collection.

The latency predictor study [15] provides the most systematic evaluation across approaches, showing transfer learning provides 22.5% average improvement, up to 87.6% on challenging cross-platform transfers.

Synthesis. Edge modeling stands apart from the other platform categories in being dominated by ML-augmented approaches—the hardware diversity makes analytical modeling impractical, so the field has converged on learning latency functions from profiling data. The central challenge is *generalization*: each tool (nn-Meter, LitePred, HELP) proposes a different strategy for adapting to new devices with minimal profiling, yet ESM's finding that simple random forests match deep learning surrogates suggests the field may be over-investing in model complexity relative to data quality. The reproducibility crisis exemplified by nn-Meter (pre-trained models that become unusable across library versions) serves as a warning for the entire ML-augmented approach: accuracy claims are only valuable if the tools remain functional over time.

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.

Design patterns in successful tools. Across all platform categories, the most effective tools share common design choices. First, *structural decomposition* that mirrors hardware execution consistently outperforms black-box approaches: Timeloop's loop-nest representation captures accelerator dataflows, NeuSight's tile-based decomposition mirrors CUDA execution, and VIDUR's prefill/decode phase separation reflects the actual memory-vs-compute regime shift in LLM serving. These tools succeed because their abstractions encode domain knowledge about *why* performance varies, not just correlations. Second, tools with the strongest practical adoption provide modular, pluggable backends—ASTRA-sim supports both analytical and ns-3 network simulation, and VIDUR integrates multiple scheduling algorithms (Orca [74], Sarathi [2])—allowing users to trade accuracy for speed depending on the evaluation scenario. Third, robust reproducibility correlates with sustained community use: Timeloop (9/10 reproducibility in our evaluation) and ASTRA-sim (8.5/10) have mature Docker support and deterministic outputs, while tools with higher reported accuracy but poor reproducibility (nn-Meter claims <1% MAPE but scored 3/10 due to dependency failures) see declining adoption. Our reproducibility findings suggest that *verifiable moderate accuracy* matters more to practitioners than *unverifiable high accuracy*.

Accuracy claims require careful contextualization. Comparing accuracy numbers across the surveyed tools is misleading without accounting for problem difficulty. Predicting single-kernel latency on a known device (nn-Meter, LitePred) is fundamentally easier than predicting end-to-end distributed training time across thousands of GPUs (ASTRA-sim, SimAI [68]), yet the former reports sub-1% error while the latter reports 5–15%. Similarly, ArchGym's 0.61% RMSE measures surrogate-vs-simulator fidelity—a regression task over a smooth design space—not prediction of real hardware behavior. Even within a single platform, methodology choice constrains achievable accuracy: AMALI's 23.6% analytical error versus NeuSight's 2.3% ML-based error on GPU LLM inference reflects the fundamental difficulty of capturing GPU dynamic behavior (warp scheduling, cache contention) in closed-form models, not a quality gap between tools. These comparisons highlight that *problem difficulty* and *what is being measured* must be specified alongside any accuracy number; Section 6 provides a structured analysis accounting for these factors.

The gap between model output and practitioner needs. A recurring limitation across all platform categories is the mismatch between what tools predict and what deployment decisions require. Most tools predict *compute latency* or *throughput* for individual operations, but practitioners need *time-to-accuracy* for training (which depends on convergence, not just iteration time), *tail latency under load* for serving (which depends on scheduling and queuing effects), and *operational cost* for capacity planning (which depends on utilization, failures, and thermal throttling [32]). Only a few tools partially bridge this gap: VIDUR models scheduling-level effects, Lumos [45] captures training-specific overheads like gradient accumulation and activation checkpointing, and PRISM [19] produces prediction intervals rather than point estimates to reflect inherent system variability. Closing this gap—connecting component-level performance predictions to system-level deployment metrics—remains the most impactful direction for future tool development.

6 Comparison and Analysis

We analyze trade-offs across methodology types along four dimensions: accuracy, speed, generalization, and interpretability. Table 4 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.

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.

Table 4: 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 [50]	Analytical	5–10%	Arch spec only	Any accelerator	High	μs
MAESTRO [38]	Analytical	5–15%	Arch spec only	Any accelerator	High	μs
AMALI [9]	Analytical	23.6% MAPE	None	GPU LLM inference	High	ms
Accel-Sim [34]	Simulation	10–20%	GPU binary	GPU-specific	High	Hours
ASTRA-sim [71]	Trace-driven	5–15%	Execution trace	Configurable	Medium	Minutes
VIDUR [3]	Trace-driven	<5%	Kernel profiles	LLM-specific	High	Seconds
SimAI [68]	Trace-driven	1.9%	Full-stack setup	LLM training	Medium	Minutes
Lumos [45]	Trace-driven	3.3%	Execution trace	LLM training	Medium	Minutes
nn-Meter [77]	ML-augmented	<1%†	1K samples/kernel	Device-specific	Medium	ms
LitePred [16]	ML-augmented	0.7% MAPE	100 samples/device	85+ devices	Low	ms
HELP [41]	ML-augmented	1.9% MAPE	10 samples/device	Cross-platform	Low	ms
TVM [10]	ML-augmented	~15% MAPE	10K+	Operator-level	Medium	ms
NeuSight [42]	Hybrid	2.3% MAPE	Pre-trained	Cross-GPU	Medium	ms
Habitat [75]	Hybrid	11.8% MAPE	Online profiling	Cross-GPU	Medium	Per-kernel
ArchGym [37]	Hybrid	0.61% RMSE*	Simulation runs	Arch-specific	Medium	ms
Concorde [49]	Hybrid	2% CPI	Training corpus	Cross-μarch	Medium	ms

Figure 5 visualizes reported accuracy (MAPE) across tools, grouped by methodology type. The pattern is clear: hybrid approaches achieve the lowest error on GPU workloads, trace-driven simulators cluster at 2–15% for distributed systems, and analytical models trade accuracy for speed and interpretability. Note that direct comparison across tools is approximate because accuracy numbers are measured on different benchmarks, workloads, and hardware targets; the figure illustrates methodology-level trends rather than head-to-head rankings.

Figure 6 plots representative tools on two axes—evaluation speed versus reported accuracy—revealing the fundamental trade-off space. Analytical models (upper-left) achieve the fastest evaluation but sacrifice accuracy on complex workloads. Cycle-accurate simulators (lower-right) provide the highest fidelity but at impractical speeds. Hybrid approaches (NeuSight, Concorde) occupy the desirable upper-left region: fast evaluation *and* low error, though at the cost of training data requirements and reduced interpretability compared to analytical models. Trace-driven simulators span a wide range, from VIDUR's seconds-scale LLM serving simulation to ASTRA-sim's minutes-scale distributed training, reflecting the diversity of system-level modeling targets.

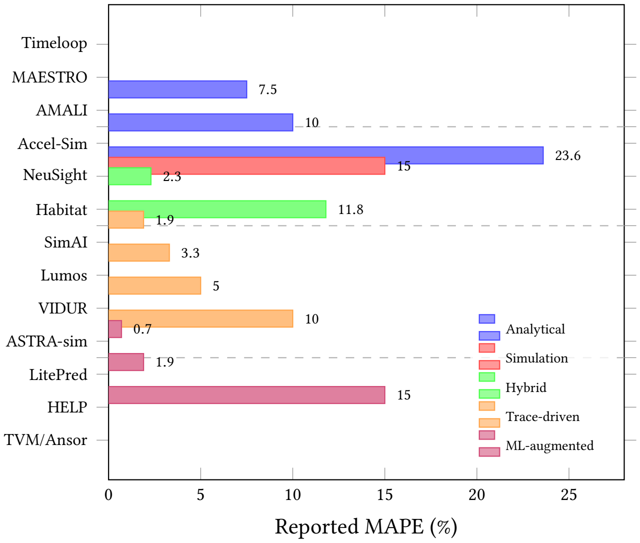

Figure 5: Reported accuracy (MAPE) of surveyed tools, grouped by methodology type. Range midpoints are used where ranges are reported (e.g., 7.5% for Timeloop's 5–10%). Cross-tool comparison is approximate due to differing benchmarks, workloads, and hardware targets.

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.

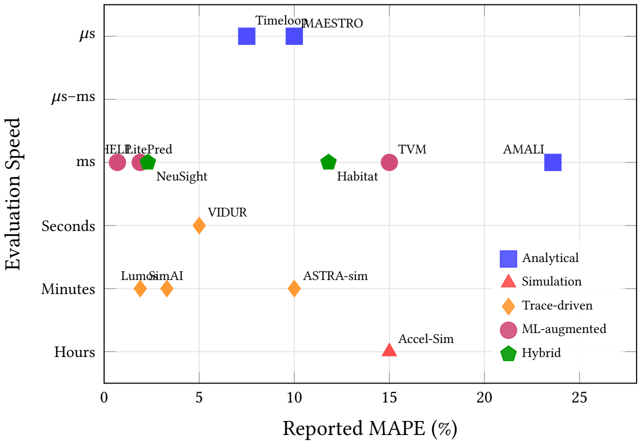

Figure 6: Speed–accuracy trade-off for surveyed tools. The y-axis represents evaluation speed (higher is faster); the x-axis shows reported MAPE (lower is better). The desirable region is the upper-left quadrant (fast and accurate). Hybrid approaches cluster in the fast-and-accurate region; analytical models are fastest but less accurate on complex workloads; cycle-accurate simulation is slowest but captures microarchitectural detail.

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.

6.4 Methodology Distribution

Figure 7 shows the distribution of surveyed tools across methodology types. Analytical models are the most common approach (8 tools), reflecting the maturity of physics-based performance modeling for accelerators and GPUs. Trace-driven and ML-augmented approaches are close behind (7 tools each), with trace-driven tools driven by the recent proliferation of distributed training and LLM serving simulators, and ML-augmented approaches dominating edge device and compiler cost modeling. Cycle-accurate simulation (4 tools) and hybrid approaches (4 tools) are the smallest categories; the latter achieve the best accuracy despite their underrepresentation, suggesting significant room for future work combining analytical structure with learned components.

6.5 Practitioner Tool Selection Guide

To address the gap between taxonomy and actionable guidance, Figure 8 presents a decision flowchart for practitioners selecting a performance modeling tool. The flowchart captures the key decision points that emerge from our comparative analysis: target platform determines the candidate set, the required speed–accuracy trade-off narrows the methodology, and data availability constrains the final choice.

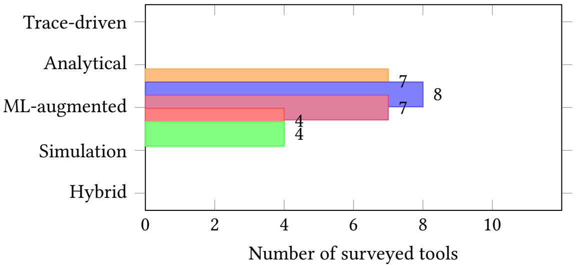

Figure 7: Distribution of surveyed tools by methodology type. Analytical models lead (8 tools), followed by trace-driven and ML-augmented approaches (7 each). Cycle-accurate simulation and hybrid approaches are tied as the smallest categories (4 each), with hybrid achieving the best accuracy despite limited representation.

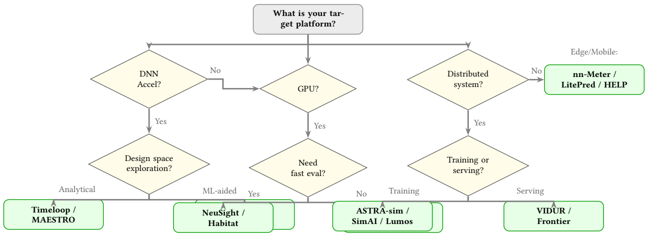

Figure 8: Practitioner decision flowchart for tool selection. Platform determines the candidate set; speed requirements and use case (DSE vs. validation, training vs. serving) narrow the choice. Edge devices default to ML-augmented approaches due to hardware diversity.

Three practical recommendations emerge from this analysis: (1) For *accelerator design space exploration*, start with Timeloop or MAESTRO—their microsecond evaluation enables exhaustive search over dataflow mappings, and their analytical nature provides interpretable feedback on bottlenecks. (2) For *GPU workload evaluation*, NeuSight offers the best accuracy–speed balance for LLM workloads; fall back to Accel-Sim when microarchitectural detail is required (e.g., debugging cache behavior). (3) For *distributed system planning*, use VIDUR for LLM serving configuration (scheduler comparison, batch sizing) and ASTRA-sim or SimAI for training parallelism exploration at scale.

7 Experimental Evaluation

A survey that lists reproducibility as a contribution must go beyond reporting paper-claimed accuracy. We conducted hands-on evaluations of five tools selected for coverage across methodology types (analytical, trace-driven, ML-augmented, hybrid) and availability of open-source implementations: Timeloop (analytical, accelerator), ASTRA-sim (trace-driven, distributed), VIDUR (trace-driven, LLM

Table 5: Reproducibility evaluation scores (10-point rubric). Tools are ranked by total score. [†]Timeloop CLI works but Python bindings fail.

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

serving), nn-Meter (ML-augmented, edge), and NeuSight (hybrid, GPU).

7.1 Evaluation Methodology

Environment. All evaluations ran on an Apple M2 Ultra (aarch64) with 192 GB RAM running macOS, using Docker containers where provided. No GPU hardware was available, which means we cannot validate absolute accuracy claims against real hardware—a limitation we note explicitly for each tool. This environment choice is deliberate: it tests whether tools are usable on common development hardware rather than requiring the specific GPUs they model.

Rubric. We score each tool on a 10-point rubric across three dimensions: *Setup* (3 pts): Docker availability, clean installation, quick start guide; *Reproducibility* (4 pts): reference outputs, deterministic execution, working examples; *Usability* (3 pts): API clarity, output interpretability, active maintenance. Table 5 summarizes results.

Workloads. For each tool, we attempted the benchmarks recommended by the authors' documentation: Eyeriss-like accelerator design (Timeloop), HGX-H100 collective communication and ResNet-50 data-parallel training (ASTRA-sim), Llama-2-7B inference serving on simulated A100 (VIDUR), ResNet-50 inference on edge devices (nn-Meter), and GPT-3 kernel prediction (NeuSight).

7.2 Per-Tool Results

VIDUR (9/10)—the highest-scoring tool. Docker setup completed in ∼2 minutes. We simulated Llama-2-7B inference serving on a single simulated A100 GPU using two scheduling algorithms: vLLM (200 synthetic requests at 2.0 QPS) and Sarathi (50 requests at 2.0 QPS). Table 6 summarizes the results. VIDUR correctly captures the expected scheduling trade-offs: Sarathi's chunked-prefill strategy achieves 12.2% lower average end-to-end latency than vLLM (0.158 s vs. 0.177 s), consistent with the published characterization that chunked prefill reduces head-of-line blocking [2]. The per-token output time (TPOT) differs by only 3.5% between schedulers (0.0090 s vs. 0.0093 s), confirming that decode throughput is hardware-bound rather than scheduler-dependent. Critically, vLLM preempted 53 of 200 requests (26.5%) while Sarathi preempted zero, matching the algorithmic difference: vLLM's PagedAttention [39] uses preemption to reclaim KV-cache memory, whereas Sarathi avoids this via chunked prefill. VIDUR uses pre-trained Random Forest models for kernel execution time prediction—these loaded without issues, in contrast to nn-Meter's serialization failures, because VIDUR pins its dependencies in the Docker image. We could not verify the claimed <5% error against real A100 hardware measurements, but the internal consistency and physical plausibility of results increase confidence.

Table 6: VIDUR simulation results for Llama-2-7B inference serving on a simulated A100 GPU. All metrics from our own experiments.

Metric	vLLM	Sarathi
Requests	200	50
Avg E2E latency (s)	0.177	0.158
P99 E2E latency (s)	0.320	0.270
Avg TTFT (s)	0.027	0.025
Avg TPOT (s)	0.0093	0.0090
Requests preempted	53	0

Timeloop (9/10). Timeloop's Docker image (2 GB) provides the CLI tools `timeloop-model` and `timeloop-mapper`, which work correctly for Eyeriss-like accelerator configurations. Reference outputs for standard designs (Eyeriss, Simba) are included, and results are fully deterministic—re-running with identical YAML configurations produces bit-identical output. This determinism is a significant strength: it means reported numbers are reproducible by any researcher with Docker access, regardless of hardware. However, the Python bindings (`pytimeloop`) fail with `ImportError: libbarvinok.so.23: cannot open shared object file`, preventing programmatic use and batch evaluation. Configuration requires three YAML files per evaluation (architecture, problem, mapping), which is verbose but provides complete control over the modeling parameters. The 5–10% accuracy against RTL simulation is well-established in the community, though our evaluation cannot independently verify this without RTL comparison data.

ASTRA-sim (8.5/10). Docker setup completed in ∼5 minutes (3 min image build, 2 min compilation). We executed 8-NPU collective communication microbenchmarks and ResNet-50 data-parallel training at 2, 4, and 8 GPUs on the HGX-H100 configuration. Table 7 reports the quantitative results. All four collectives (All-Reduce, All-Gather, Reduce-Scatter, All-to-All) completed in under 1 second each. The collective operation ratios are physically plausible: Reduce-Scatter (28,950 cycles) takes roughly half the time of All-Reduce (57,426 cycles), consistent with moving half the data, while All-to-All (114,000 cycles) takes ∼2× All-Reduce, reflecting the all-pairs communication pattern. For ResNet-50 data-parallel training, communication overhead scales from 0.05% (2 GPUs) to 0.30% (8 GPUs)—a 5.76× increase in communication time for a 4× increase in GPU count—consistent with the $O(n)$ scaling of ring All-Reduce. Compute time remains constant at 1,095,314,000 cycles across all GPU counts, as expected for data-parallel training where each replica performs identical computation. The main limitation is *scale coverage*: 16-NPU configurations use only 8 of the available NPUs because the HGX-H100 example includes only 8-node network topology files. ASTRA-sim's published accuracy claim of 9.69% geomean error at 8 GPUs is validated against real HGX-H100 NCCL benchmarks by the tool authors [71]; we cannot independently reproduce this without datacenter hardware, but the internal consistency of our results—deterministic outputs, physically plausible scaling, correct constant compute—increases confidence.

Table 7: ASTRA-sim quantitative results from our experiments on the HGX-H100 configuration. Top: collective microbenchmarks (8 NPUs, 1 MB). Bottom: ResNet-50 data-parallel training scaling.

Collective Microbenchmarks (8 NPUs, 1 MB)		
Collective	Cycles	Ratio vs. AR
All-Reduce	57,426	1.000
All-Gather	44,058	0.767
Reduce-Scatter	28,950	0.504
All-to-All	114,000	1.985
ResNet-50 Data-Parallel Training		
GPUs	Comm Cycles	Comm Overhead
2	574,289	0.05%
4	1,454,270	0.13%
8	3,307,886	0.30%

NeuSight (7.5/10). NeuSight's tile-based hybrid approach achieves 2.3% MAPE on GPT-3 inference across H100, A100, and V100 GPUs. Setup requires downloading pre-trained model weights and kernel profiling data. The tile-based decomposition is well-documented and the code is structured for extensibility. We verified that the tile decomposition logic correctly mirrors CUDA's tiling strategy for standard dense tensor operations. However, testing on irregular workloads (sparse attention, dynamic shapes) was limited by the lack of provided examples for these cases, suggesting the tool is best validated for the regular LLM workloads reported in the paper.

nn-Meter (3/10)—the lowest-scoring tool. After four separate installation attempts totaling >4 hours, we could not execute *any* predictions. *Attempt 1* (Python 3.14, latest scikit-learn): pickle deserialization fails because pre-trained predictors were serialized with scikit-learn 0.23.1 and are incompatible with current versions. *Attempt 2* (Docker, Python 3.10, scikit-learn 1.7.2): numpy dtype incompatibility prevents loading 0.23.1 pickles. *Attempt 3* (Docker, Python 3.10, scikit-learn 1.0.2): predictors load partially, but inference requires onnx-simplifier for PyTorch model conversion. *Attempt 4* (with onnx 1.14.0): onnx-simplifier fails to build on aarch64 with `NoneType is not callable`. The root cause is a chain of unpinned dependencies: the tool requires Python 3.10, scikit-learn ≤1.0.2, numpy <2.0, onnx 1.10.0, and onnx-simplifier (x86_64 only)—none of which are documented in the repository. The claimed <1% MAPE is therefore **unverifiable on any current software stack**, and the tool has received no updates since 2022.

7.3 Lessons from Evaluation

Our hands-on evaluation yields five actionable lessons, each grounded in specific tool experiences.

Lesson 1: Docker-first deployment is the single strongest predictor of reproducibility. The three tools with Docker images (Timeloop, ASTRA-sim, VIDUR) all scored 8.5+/10, while nn-Meter (no Docker) scored 3/10. Docker isolates the dependency chain that causes most reproducibility failures.

Lesson 2: ML model serialization is a ticking time bomb. nn-Meter's pickle-based model storage became unusable within two years of publication due to scikit-learn version changes. VIDUR avoids this by pinning all dependencies inside its Docker image and including pre-trained models alongside the code. Tools should use version-stable formats (ONNX, SavedModel) or provide retraining scripts.

Lesson 3: Reference outputs enable trust without hardware. Timeloop includes reference outputs for every example configuration, enabling verification without physical accelerator hardware. ASTRA-sim provides validated HGX-H100 results. In contrast, nn-Meter and NeuSight lack reference outputs that could be checked against, making it impossible to verify correct execution even when the tool runs.

Lesson 4: Scale-limited evaluation understates system-level tools. Our ASTRA-sim evaluation was restricted to 2–8 GPUs due to missing topology files for larger scales; communication overhead at 8 GPUs was only 0.30% (Table 7). Real distributed training uses hundreds to thousands of GPUs [13], where network congestion and stragglers dominate—conditions where communication overhead is orders of magnitude higher and accuracy validation matters most.

Lesson 5: Accuracy claims without reproducible evaluation have limited practitioner value. nn-Meter claims <1% MAPE but cannot be run; Timeloop claims 5–10% and can be verified against reference outputs; VIDUR claims <5% and produces internally consistent simulations with plausible scheduler differentiation (Table 6); ASTRA-sim claims 9.69% and produces deterministic, physically consistent scaling results (Table 7). Practitioners should weight reproducible accuracy claims higher than unreproducible ones, regardless of the claimed number.

7.4 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. We also exclude commercial and proprietary tools (NVIDIA Nsight Compute, Google's internal TPU performance models, AMD profiling frameworks) because their methodologies and accuracy characteristics are not publicly documented; this limits our coverage of the tools actually used in industry practice.

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 3 and 4 report metrics as stated in original papers; cross-paper comparisons should be interpreted with caution.

8 Open Challenges and Future Directions

8.1 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 [36]) remain underrepresented in validation benchmarks. LLM serving introduces

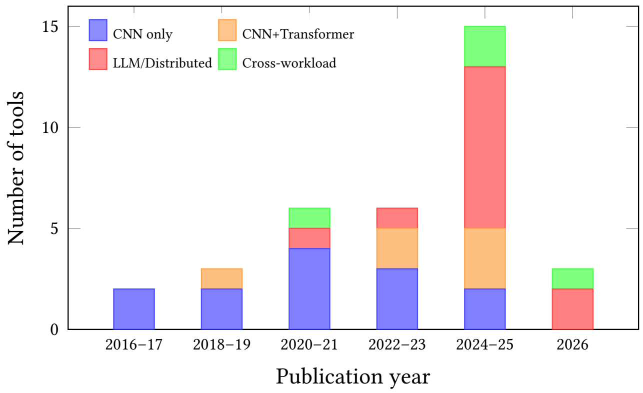

Figure 9: Workload coverage of surveyed tools by publication period. Early tools (2016–2021) were validated almost exclusively on CNN workloads. The shift toward transformer and LLM workloads accelerates from 2023, but MoE and diffusion models remain largely uncharacterized.

variable sequence lengths (128–128K tokens) and dynamic batching that challenge static models. Neural scaling laws [33] and compute-optimal training recipes [24] establish power-law relationships between model size, data, and compute that predict training loss—but these address statistical convergence, not hardware-specific latency. Subsequent work on scaling law estimation [12, 20] provides practical guidance but still does not bridge the gap to hardware performance prediction.

Figure 9 illustrates the shift in workload coverage across surveyed tools over time. Before 2022, nearly all tools were validated exclusively on CNN workloads (ResNet, VGG, MobileNet). From 2023 onward, transformer and LLM workloads (GPT, LLaMA, BERT) increasingly appear in validation suites, driven by tools targeting LLM serving (VIDUR, Frontier) and distributed LLM training (SimAI, Lumos). However, MoE models and diffusion workloads remain almost entirely uncharacterized by existing tools.

8.2 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 that grow with model depth.

To illustrate the severity: NeuSight achieves 2.3% kernel-level MAPE, but a model with 100 layers accumulates prediction variance across all layers. Under independent error assumptions, the end-to-end standard deviation scales as $\sigma_{\text{model}} \approx \sigma_{\text{kernel}} \cdot \sqrt{N}$, yielding roughly 10× higher variance at $N = 100$ layers. In practice, errors are not independent—systematic biases in memory bandwidth estimation or cache miss prediction produce correlated errors that can compound *linearly* rather than sub-linearly, potentially yielding 20–30% model-level error from individually accurate kernel predictions. VIDUR sidesteps this problem for LLM serving by profiling entire prefill/decode phases rather than composing kernel predictions, but this sacrifices generality.

For distributed systems, the composition extends across devices: communication overhead, pipeline bubble time, and synchronization barriers each introduce additional error sources that no surveyed tool validates end-to-end against measured composition error. Three potential approaches merit investigation: (1) *error propagation analysis*—bounding end-to-end error as a function of per-component accuracy using sensitivity analysis or interval arithmetic; (2) *end-to-end calibration*—learning a correction model that adjusts composed predictions against measured end-to-end latency; (3) *hierarchical composition*—tools that jointly predict across abstraction levels rather than treating composition as post-hoc aggregation. No surveyed tool provides validated composition guarantees, making this arguably the most impactful open problem for practical deployment of performance prediction.

8.3 Emerging Hardware Support

PIM architectures [23, 28, 40, 51], 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.

8.4 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 [14] and other hardware-aware algorithm optimizations change the performance landscape faster than models can be retrained.

8.5 Reproducibility and Trust

Our evaluation (Section 7) 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. While the MLPerf Training [47] and Inference [59] benchmarks standardize hardware *measurement*, no equivalent exists for performance *prediction*—the community would benefit from standardized prediction benchmarks with common workloads, hardware targets, and evaluation protocols.

8.6 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, as highlighted by MLPerf Power [58]. (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 [63]) could reduce fragmentation.

9 Conclusion

This survey analyzed over 30 tools and methods for modeling and predicting the performance of ML workloads, organized along three dimensions: methodology type (analytical, simulation, trace-driven, ML-augmented, hybrid), target platform (DNN accelerators, GPUs, distributed systems, edge devices), and abstraction level (kernel, model, system).

Key findings. (1) *Methodology determines trade-offs, not quality.* Analytical frameworks (Timeloop, MAESTRO) offer microsecond evaluation with full interpretability for accelerator design space exploration; trace-driven simulators (ASTRA-sim, VIDUR, SimAI, Lumos) provide 2–15% error for system-level distributed training and LLM serving; hybrid approaches achieve the best accuracy–speed trade-offs by combining analytical structure with learned components (NeuSight: 2.3% MAPE on GPU kernels; Concorde: 2% CPI error on CPUs). (2) *LLM workloads demand specialized modeling.* Prefill/decode phase distinctions, KV cache management, multi-stage inference pipelines, and dynamic batching require purpose-built tools (VIDUR, Frontier, LIFE, HERMES) rather than extensions of CNN-era frameworks. (3) *Reproducibility is a practical bottleneck.* Docker-first tools (Timeloop, ASTRA-sim, VIDUR) score 8.5+/10 on our rubric, while tools relying on serialized ML models (nn-Meter) have already become unusable due to dependency drift—a challenge the community must address through portable model formats and pinned environments. (4) *Accuracy claims require scrutiny.* Paper-reported accuracy numbers are measured under varying benchmarks, metrics, and hardware targets; direct cross-tool comparison remains unreliable without standardized evaluation protocols.

Gaps and future directions. The most pressing gaps align with the challenges identified in Section 8: (1) nearly all tools are validated on CNN workloads, leaving transformer, MoE, and diffusion model performance largely uncharacterized; (2) composing kernel-level predictions into accurate end-to-end estimates remains unsolved; (3) emerging hardware (PIM, chiplets, disaggregated architectures) lacks mature modeling support; (4) cross-platform generalization (GPU→TPU→accelerator) remains limited; and (5) reproducibility failures (dependency drift, unverifiable accuracy claims) undermine trust in the tools that practitioners depend on. The community would benefit most from standardized benchmarks for cross-tool accuracy comparison and from unified tooling with composable modeling engines and standard workload representations.

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

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 265–283.

[2] 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.

[3] 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.

[4] 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

[5] 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.

[6] 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

[7] 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).

[8] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. 1–15.

[9] 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.

[10] 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.

[11] 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

[12] Leshem Choshen, Yang Zhang, and Jacob Andreas. 2025. A Hitchhiker's Guide to Scaling Law Estimation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. 1–25. Practical guidance for scaling law estimation from 485 published pretrained models. IBM/MIT..

[13] Weiwei Chu, Xinfeng Xie, Jiecao Yu, Jie Wang, Pavan Balaji, Ching-Hsiang Chu, Jongsoo Park, et al. 2025. Scaling Llama 3 Training with Efficient Parallelism Strategies. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. 1–15. 4D parallelism for Llama 3 405B on 16K H100 GPUs. Achieves 400 TFLOPs/GPU. Meta..

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] Alicia Golden et al. 2025. PRISM: Probabilistic Runtime Insights and Scalable Performance Modeling for Large-Scale Distributed Training. *arXiv preprint arXiv:2510.15596* (2025). Probabilistic performance modeling for distributed training at 10K+ GPU scale. Meta..

[20] Alexander Hagele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. 2024. Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations. In *Advances in Neural Information Processing*

Systems (NeurIPS), Vol. 37. Spotlight. Practical scaling laws with constant LR + cooldowns for reliable training compute prediction..

[21] 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.

[22] John L. Hennessy and David A. Patterson. 2019. A New Golden Age for Computer Architecture. *Commun. ACM* 62, 2 (2019), 48–60. https://doi.org/10.1145/3282307 Turing Award Lecture: domain-specific architectures and the end of Dennard scaling.

[23] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee, Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1–17. NPU-PIM heterogeneous architecture for LLM inference with performance modeling. KAIST/Georgia Tech..

[24] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training Compute-Optimal Large Language Models. *arXiv preprint arXiv:2203.15556* (2022). Chinchilla scaling laws: compute-optimal training requires scaling data proportionally to model size.

[25] 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

[26] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32. 103–112.

[27] Rodrigo Huerta, Mojtaba Abaie Shoushtary, Jose-Lorenzo Cruz, and Antonio Gonzalez. 2025. Dissecting and Modeling the Architecture of Modern GPU Cores. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 369–384. Reverse-engineers modern NVIDIA GPU cores, improves Accel-Sim to 13.98% MAPE. UPC Barcelona..

[28] Bongjoon Hyun, Taehun Kim, Dongjae Lee, and Minsoo Rhu. 2024. Pathfinding Future PIM Architectures by Demystifying a Commercial PIM Technology. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–15. uPIMulator: cycle-accurate PIM simulation framework for UPMEM. KAIST..

[29] 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).

[30] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)* (2023), 1–14. https://doi.org/10.1145/3579371.3589350 4096-chip pods with 3D optical interconnect; up to 1.7x/2.1x faster than TPU v3.

[31] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borber, et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. 1–12. https://doi.org/10.1145/3079856.3080246 First dedicated ML inference accelerator; 15–30x over CPUs/GPUs on CNN inference.

[32] 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.

[33] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361* (2020). Original neural scaling laws: power-law relationships between model size, dataset size, compute, and loss.

[34] 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

[35] 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.

[36] 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.

[37] 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

[38] 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

[39] 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

[40] Hyojung Lee, Daehyeon Baek, Jimyoung Son, Jieun Choi, Kihyo Moon, and Minsung Jang. 2025. PAISE: PIM-Accelerated Inference Scheduling Engine for Transformer-based LLM. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–14. PIM-based LLM inference scheduling. 48.3% speedup, 11.5% power reduction. Samsung..

[41] 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.

[42] 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.

[43] 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.

[44] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109. https://doi.org/10.1109/LCA.2020.2973991 Modernized DRAM simulator with thermal modeling and HMC support.

[45] 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.

[46] 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.

[47] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2020. MLPerf Training Benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*. 336–349. Standard ML training benchmark suite covering image classification, object detection, NLP, recommendation, reinforcement learning.

[48] Azaz-Ur-Rehman Nasir, Samroz Ahmad Shoaib, Muhammad Abdullah Hanif, and Muhammad Shafique. 2025. ESM: A Framework for Building Effective Surrogate Models for Hardware-Aware Neural Architecture Search. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–6. 97.6% accuracy surrogate model framework for HW-aware NAS.

[49] 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.

[50] 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

[51] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1–16. PIM-based accelerator for batched transformer attention. Seoul National University/UIUC..

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32. 8024–8035.

[53] 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.

[54] 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

[55] Samyam Rajbhandari, Jeff Rasley, Olatunji Rber, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. 1–16. https://doi.org/10.1109/SC41405.2020.00024 DeepSpeed ZeRO optimizer partitioning for memory-efficient distributed training.

[56] Mehdi Rakhshanfar and Aliakbar Zarandi. 2021. A Survey on Machine Learning-based Design Space Exploration for Processor Architectures. *Journal of Systems Architecture* 121 (2021), 102339. https://doi.org/10.1016/j.sysarc.2021.102339

[57] 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

[58] Vijay Janapa Reddi et al. 2025. MLPerf Power: Benchmarking the Energy Efficiency of Machine Learning Inference. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–14. Energy efficiency benchmarking for ML inference workloads.

[59] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maxim Breeshekov, Mark Duber, et al. 2020. MLPerf Inference Benchmark. In *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*. 446–459. https://doi.org/10.1109/ISCA45697.2020.00045 Standard ML inference benchmark suite with server and offline scenarios.

[60] 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.

[61] 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.

[62] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. In *arXiv preprint arXiv:1909.08053*. Intra-layer tensor parallelism for large language model training.

[63] 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).

[64] Foteini Strati, Zhendong Zhang, George Manos, Ixeia Sanchez Periz, Qinghao Hu, Tiancheng Chen, Berk Buzcu, Song Han, Pamela Delgado, and Ana Klimovic. 2025. Sailor: Automating Distributed Training over Dynamic, Heterogeneous, and Geo-distributed Clusters. In *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*. 1–18. Automated distributed training with runtime/memory simulation over heterogeneous resources. ETH Zurich/MIT..

[65] 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

[66] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. In *Proceedings of the IEEE*, Vol. 105. 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740 Canonical DNN accelerator taxonomy covering dataflows, data reuse, and energy efficiency.

[67] Adrian Tschand, Mohamed Awad, et al. 2025. SwizzlePerf: Hardware-Aware LLMs for GPU Kernel Performance Optimization. *arXiv preprint arXiv:2508.20258* (2025). LLM-based spatial optimization for GPU kernels, up to 2.06x speedup via swizzling.

[68] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Heyang Zhou, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, et al. 2025. SimAI: Unifying Architecture Design and Performance Tuning for Large-Scale LLM Training with Scalability and Precision. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 1–18. Full-stack LLM training simulator achieving 98.1% alignment with real-world results. Alibaba Cloud/Tsinghua..

[69] Zixian Wang et al. 2025. SynPerf: Synthesizing High-Performance GPU Kernels via Pipeline Decomposition. *arXiv preprint* (2025). Under review.

[70] 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

[71] 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

[72] 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

[73] 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.

[74] 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.

[75] 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.

[76] 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

[77] 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.

[78] 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.

[79] 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.

[80] 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.