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

This survey analyzes 22 performance modeling tools from 53 papers (2016–2026), covering analytical models, trace-driven simulators, and ML-augmented hybrids for DNN accelerators, GPUs, distributed training, and LLM inference. We organize the literature by methodology type, target platform, and abstraction level, identifying a temporal validation lag and finding that hybrid approaches achieve the best accuracy-speed trade-offs. Hands-on reproducibility evaluations show Docker-first tools remain reproducible while those relying on serialized ML models become unusable. We identify open challenges in cross-workload generalization, error composition, and emerging architecture support.

Keywords

ML workload performance prediction, DNN accelerator modeling, GPU simulation, distributed training simulation, LLM inference serving, design space exploration, survey

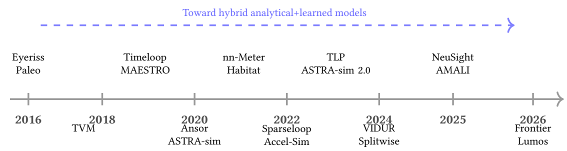

Figure 1: Evolution of performance modeling tools (2016–2026). Early analytical frameworks gave way to systematic accelerator modeling and distributed training simulation. Recent work targets LLM-specific and hybrid approaches.

1 Introduction

Machine learning workloads have become the dominant consumers of compute across datacenters and edge devices. Training and inference for CNNs, transformers, mixture-of-experts models, and LLMs demand hardware ranging from Google's TPU [34, 35] to custom accelerators, creating a heterogeneous landscape where architects must predict performance before committing to costly hardware decisions.

The shift toward domain-specific architectures [25] makes performance prediction both more important and more difficult. Design space exploration, parallelization selection, and hardware-software co-design all require fast, accurate performance models—yet ML workloads pose unique challenges: diverse computational patterns (dense matrix operations, sparse accesses, communication-bound collectives) across GPUs, TPUs, custom accelerators, and multi-device clusters.

A rich ecosystem of modeling tools has emerged. Analytical models (Timeloop [57], MAESTRO [43]) evaluate in microseconds with 5–15% error. Trace-driven simulators (ASTRA-sim [83], VIDUR [3]) replay execution traces for system-level modeling. Hybrid approaches (NeuSight [48]) combine analytical structure with learned components. Yet no comprehensive survey organizes these methods for the practitioner who must select a tool for a specific task. Existing surveys focus on ML *techniques* for modeling [75] or specific hardware [57]; this survey fills that gap with a methodology-centric view that yields new architectural insights.

MICRO 2026, Austin, TX, USA
2026.

We make the following contributions:

- A **methodology-centric taxonomy** organizing tools along three dimensions: methodology type, target platform, and abstraction level, with a coverage matrix identifying explicit research gaps (e.g., no trace-driven tools for accelerators, no hybrid tools for distributed systems).
- A **cross-methodology architectural analysis** revealing why structural decomposition aligned with hardware execution boundaries (loop nests for systolic arrays, tiles for GPU SMs, phases for serving) consistently outperforms methodology-agnostic approaches—an insight that cuts across subdomain boundaries and provides concrete design principles for future tools.
- **Hands-on reproducibility evaluation** of five representative tools, demonstrating that deployment methodology (Docker-first vs. serialized ML models) is a stronger predictor of usability than reported accuracy, with implications for tool design.
- An **error composition analysis** characterizing how kernel-level prediction errors propagate through the model-to-system abstraction stack, identifying the uncaptured inter-kernel overheads (launch latency, memory allocation, synchronization) that dominate the gap between kernel and end-to-end accuracy.

Figure 1 illustrates the evolution of performance modeling tools from early analytical frameworks to modern hybrid approaches.

2 Survey Methodology

We searched ACM Digital Library, IEEE Xplore, Semantic Scholar, and arXiv using terms related to ML performance modeling, with backward/forward citation tracking from seminal works. Target venues include architecture (MICRO, ISCA, HPCA, ASPLOS), systems (MLSys, OSDI, SOSP, NSDI), and related (NeurIPS, MobiSys, DAC, ISPASS). Papers must propose or evaluate a tool for predicting ML workload performance with quantitative evaluation; we exclude non-performance tasks and general-purpose workloads. From 287 initial candidates, title/abstract screening yielded 118 papers; full-text review reduced the set to 53 that met all criteria, supplemented by 12 foundational works for context. We cover 2016–2026 and classify each paper by *methodology type* (analytical, simulation, trace-driven, ML-augmented, hybrid), *target platform*, and *abstraction level* (kernel, model, system).

Related surveys and scope boundaries. Prior surveys address adjacent topics: Rakhshanfar and Zarandi [65] survey ML for processor DSE; Sze et al. [76] treat DNN hardware design (the foundation for Timeloop/MAESTRO); simulators such as GPGPU-Sim [4], gem5 [6], and SST [68] have been extensively used as validation targets in the performance modeling literature; and MLPerf [53, 67] standardizes *measurement* rather than *prediction*. Early ML accelerator modeling (2014–2018) established foundational approaches: DianNao [11] introduced analytical dataflow modeling for dedicated accelerators, Eyeriss [13] systematized row-stationary dataflow analysis, and Paleo [61] pioneered layer-wise analytical estimation. The closest prior work, Dudziak et al. [17], compares edge device predictors for NAS; we broaden to the full landscape.

Proprietary and vendor tools. NVIDIA's Nsight Compute [56] and Nsight Systems are the most widely-used GPU profiling tools in practice; Google's internal TPU models underpin production scheduling but are undocumented. We exclude these from evaluation as they cannot be independently reproduced, though surveyed tools frequently validate against Nsight Compute data.

Compiler cost models and capacity planning. Beyond TVM/Ansor/TLP, relevant compiler models include Halide's autoscheduler [63] (pioneered learned cost models), MLIR-based cost models [45], and Triton's [77] heuristic GPU kernel cost model. At the system level, Pollux [62] and Sia [33] use performance models for cluster scheduling and capacity planning—a distinct use case (optimizing workload placement) that shares modeling techniques with our surveyed tools.

This survey differs from all prior work by spanning the full methodology spectrum across all major platforms with reproducibility evaluation.

3 Background

3.1 ML Workload Characteristics

ML workloads are expressed as computation graphs whose operator shapes are statically known and amenable to analytical modeling. Frameworks such as PyTorch [59] and TensorFlow [1] compile these graphs for execution, though MoE and dynamic inference introduce input-dependent control flow. Performance depends on tensor-to-memory mapping (dataflow, tiling), KV cache management for LLM inference [44], and at scale, compute–memory–network interactions across data, tensor, pipeline, and expert parallelism [15]. LLM inference splits into compute-bound prefill and memory-bound decode phases [60], both modeled under batched serving [2, 85]. Foundation model training introduces additional modeling challenges: long-context attention with quadratic memory scaling, activation checkpointing that trades compute for memory, and mixed-precision training where numerical format affects both speed and convergence [15].

3.2 Modeling Methodologies

We classify approaches into five categories. **Analytical models** express performance as closed-form functions (e.g., the roofline model [82]), offering microsecond evaluation but requiring per-architecture derivation. **Cycle-accurate simulators** (GPGPU-Sim [4], Accel-Sim [38]) achieve high fidelity at 1000–10000× slowdown, serving primarily as validation oracles for the high-level methods that are the focus of this survey. **Trace-driven simulators** (ASTRA-sim [83], VIDUR [3]) trade fidelity for orders-of-magnitude speedup. **ML-augmented approaches** learn from profiling data (nn-Meter [88]) but may not generalize beyond training distributions. **Hybrid approaches** combine analytical structure with learned components (NeuSight [48], Habitat [86]), aiming to balance accuracy, speed, and interpretability. Accuracy metrics—MAPE, RMSE, and rank correlation—vary across the literature, limiting direct comparison (Section 6); ground-truth relies on hardware counters (PAPI [7], LIKWID [78]) or vendor profilers [56].

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, identifying a temporal validation lag: tools published during the CNN era naturally validated on CNN workloads, while post-2023 tools increasingly target transformers and LLMs.

Table 1 provides a unified view combining the coverage matrix (number of surveyed tools per methodology–platform cell) with trade-off profiles, with empty cells highlighting research gaps. The dominant pairings are: analytical models for accelerators, cycle-accurate simulation for GPUs/CPUs, trace-driven simulation for distributed systems, and ML-augmented approaches for edge devices.

Table 1 reveals three structural gaps: (1) trace-driven *execution replay* simulation (as distinct from instruction-trace-driven cycle-accurate simulation such as Accel-Sim) is used exclusively for distributed systems; (2) edge devices are served only by ML-augmented approaches, lacking hybrid alternatives; (3) no ML-augmented tool targets distributed systems directly. Methodologies cluster into two speed regimes: sub-millisecond (analytical, ML-augmented, hybrid) for DSE, and minutes-to-hours (simulation, trace-driven) for validation.

Figure 2 illustrates how tools from different methodology types compose: analytical engines provide fast base estimates, ML components learn residual corrections, and trace-driven simulators orchestrate system-level execution.

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 Table 1; Section 5 provides detailed per-tool analysis.

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

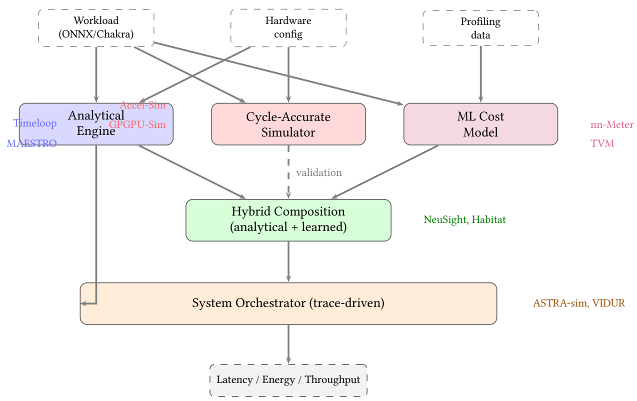

Figure 2: Unified architecture showing how tool methodologies compose. Analytical engines and ML cost models feed into hybrid approaches, while system-level orchestrators (trace-driven) assemble component predictions into end-to-end estimates. Cycle-accurate simulators primarily serve as validation oracles.

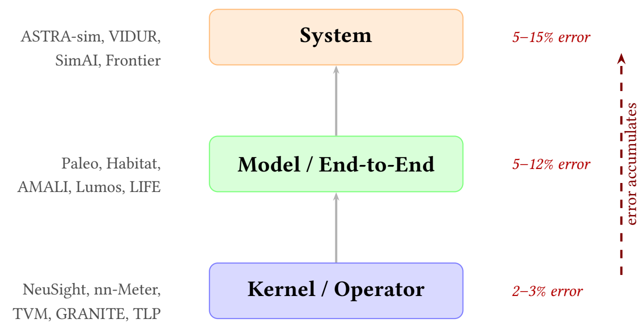

Figure 3: Abstraction level hierarchy and the composition problem. Tools operate at one of three levels; composing predictions across levels accumulates error. Error ranges are representative values from surveyed papers.

Analytical models (Timeloop [57]: 5–10% vs. RTL; MAESTRO [43]; Sparseloop [84]; AMALI [10]) provide microsecond evaluation and full interpretability but require per-architecture derivation (AMALI's 23.6% MAPE illustrates GPU dynamic effects). **Cycle-accurate simulators** (GPGPU-Sim [4], Accel-Sim [38]: 0.90–0.97 IPC; PyTorchSim [39]) are impractical for DSE at 1000–10000× slowdown [4, 38]. **Trace-driven simulators** (ASTRA-sim [83]: 5–15%; VIDUR [3]: <5%; SimAI [80]; Frontier [20]) replay execution traces for system-level modeling. **ML-augmented models** (nn-Meter [88]; LitePred [18]; HELP [47]; TVM [12]/Ansor [89]) learn from profiling data but risk *silent distribution shift.* **Hybrid** approaches (NeuSight [48]; Habitat [86]; ArchGym [42]) combine analytical priors with learned corrections [17].

4.2 Secondary Axes: Platform and Abstraction Level

Platform constrains methodology: **accelerators** use analytical models; **GPUs** span all types; **distributed systems** require trace-driven simulation; **edge devices** use ML-augmented approaches; **CPUs** [55, 75] are least studied. Abstraction level determines composition errors (Figure 3): kernel-level tools achieve 2–3% error, model-level 5–12%, and system-level 5–15%, with errors propagating through the chain.

4.3 Workload Coverage

Table 2 characterizes the workload types on which each tool has been validated, revealing a temporal validation lag rather than a methodological bias: tools published during the CNN-dominant era (2016–2022) naturally validated on the workloads of their time, while post-2023 tools increasingly target transformers and LLMs.

Figure 4 quantifies this temporal validation lag: of the 14 surveyed tools, 9 (64%) include CNN validation, reflecting the dominance of CNNs when those tools were published. Critically, the lag is closing—post-2023 tools (VIDUR, Frontier, Lumos, SimAI) validate exclusively on transformers/LLMs—but emerging workloads remain uncovered: **no surveyed tool has been validated on diffusion models or dynamic inference workloads** [40], only Frontier [20] has validated MoE support, and no single tool offers validated transformer prediction across the full kernel-to-system stack. The practical consequence: practitioners working with frontier workloads must accept unvalidated predictions, collect their own validation data, or 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	Transformer	LLM Train	MoE	Diff.
Timeloop	✓	○	—	—	—
MAESTRO	✓	—	—	—	—
NeuSight	✓	✓	—	—	—
Habitat	✓	—	—	—	—
AMALI	—	✓	—	—	—
ASTRA-sim	✓	○	✓	—	—
VIDUR	—	✓	—	—	—
SimAI	—	—	✓	—	—
Lumos	—	—	✓	—	—
Frontier	—	✓	—	✓	—
nn-Meter	✓	—	—	—	—
LitePred	✓	—	—	—	—
HELP	✓	—	—	—	—
TVM/Ansor	✓	○	—	—	—

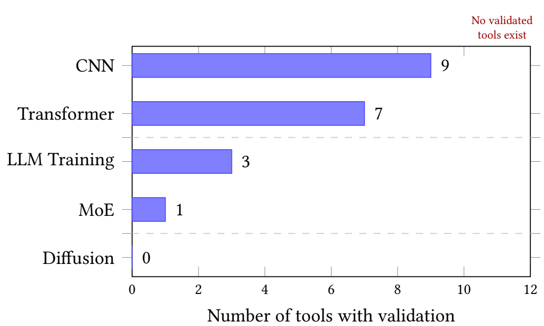

Figure 4: Workload validation coverage across surveyed tools. CNN validation reflects the temporal publication distribution (most tools published 2016–2022), while MoE and diffusion models—dominant only since 2023—have minimal or no validated prediction tools.

5 Survey of Approaches

This section surveys performance modeling tools for ML workloads, organized by target platform, examining modeling challenges, available tools, and their strengths and limitations. Table 3 provides a comprehensive comparison.

5.1 DNN Accelerator Modeling

The analytical tractability of DNN accelerator modeling stems from the regularity of computation [76], building on early characterization pioneered by DianNao [11]. A convolution layer maps to a seven-deep nested loop over batch, output channel, input channel, and spatial dimensions; Timeloop [57] enumerates mappings of these loops to a spatial-temporal hardware hierarchy, computing data reuse at each memory level as the ratio of loop bounds. This exhaustive search finds the optimal dataflow in microseconds (5–10% error, 2000× speedup) because the search space, though combinatorially large, admits efficient pruning: any mapping that exceeds a memory level's capacity is immediately discarded. MAESTRO [43] achieves similar modeling with a more compact "data-centric" representation that specifies which data dimension is stationary at each level, trading enumeration completeness for specification simplicity—but sacrificing Timeloop's ability to model per-PE utilization, explaining why Timeloop achieves tighter error bounds on architectures with irregular PE arrays. SCALE-Sim [70] complements both by providing cycle-accurate systolic array simulation for validation. Sparseloop [84] extends Timeloop's analysis to sparse tensors by introducing format-specific access count models for compression formats (CSR, bitmap)—the key challenge being that sparse data access patterns depend on the data values, requiring statistical or format-aware modeling rather than purely geometric analysis. PyTorchSim [39] integrates PyTorch 2 with NPU simulation but lacks real-hardware validation; ArchGym [42] connects ML surrogates to simulators (0.61% RMSE vs. simulator, not hardware). Accelerator modeling is the most mature subdomain, with Timeloop as the de facto DSE standard. The key gap is silicon validation; emerging PIM tools [26, 31, 46, 58] also lack hardware validation.

5.2 GPU Performance Modeling

GPUs dominate ML training and inference, requiring models for SIMT execution, warp scheduling, memory coalescing, and occupancy effects.

Cycle-accurate simulation. GPGPU-Sim [4] and Accel-Sim [38] achieve 0.90–0.97 IPC correlation but at 1000–10000× slowdown; reverse-engineering [30] improved Accel-Sim to 13.98% MAPE. These simulators integrate with memory subsystem models—from DRAMSim2 [69] and Ramulator [41] to their modern successors DRAMSim3 [50] and Ramulator 2.0 [52]—for accurate DRAM timing, critical for memory-bound LLM inference.

Analytical and hybrid models. AMALI [10] models GPU performance through memory hierarchy analysis (L1, L2, HBM data movement volumes); the roofline model [82] provides upper bounds, with recent LLM-specific extensions [32]. NeuSight [48] achieves 2.3% MAPE on GPT-3 kernels by decomposing each kernel into *tiles* corresponding 1:1 with CUDA thread blocks. This abstraction succeeds because GPU scheduling is tile-based: each SM's execution time depends on arithmetic intensity, shared memory footprint, and register pressure—all locally measurable per tile. By profiling representative tiles and extrapolating via occupancy, NeuSight captures memory bandwidth saturation and L2 cache pressure without modeling warp scheduling details. AMALI's whole-kernel model misses these effects by averaging data movement over the entire kernel, losing per-SM occupancy information. Habitat [86] achieves 11.8% cross-GPU transfer via wave scaling based on SM count and memory bandwidth ratios.

The accuracy disparity reflects a fundamental distinction: accelerator execution is deterministic (loop nests fully determine data movement), while GPUs introduce warp scheduling, memory

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 [57]	NPU	A	Latency/Energy	5–10%	μs	Loop-nest DSE
MAESTRO [43]	NPU	A	Latency/Energy	5–15%	μs	Data-centric directives
Sparseloop [84]	NPU	A	Sparse tensors	5–10%	μs	Compression modeling
PyTorchSim [39]	NPU	S	Cycle-accurate	N/A‡	Hours	PyTorch 2 integration
ArchGym [42]	Multi	H	Multi-objective	0.61%*	ms	ML-aided DSE
GPU Performance Modeling						
Accel-Sim [38]	GPU	S	Cycle-accurate	10–20%	Hours	SASS trace-driven
GPGPU-Sim [4]	GPU	S	Cycle-accurate	10–20%	Hours	CUDA workloads
AMALI [10]	GPU	A	LLM inference	23.6%	ms	Memory hierarchy
NeuSight [48]	GPU	H	Kernel/E2E latency	2.3%	ms	Tile-based prediction
Habitat [86]	GPU	H	Training time	11.8%	Per-kernel	Wave scaling
Distributed Training and LLM Serving						
ASTRA-sim [83]	Distributed	T	Training time	5–15%	Minutes	Collective modeling
SimAI [80]	Distributed	T	Training time	1.9%	Minutes	Full-stack simulation
Lumos [51]	Distributed	T	LLM training	3.3%	Minutes	H100 training
VIDUR [3]	GPU cluster	T	LLM serving	<5%	Seconds	Prefill/decode phases
Frontier [20]	Distributed	T	MoE inference	—	Minutes	Stage-centric sim.
TrioSim [49]	Multi-GPU	T	DNN training	N/A‡	Minutes	Lightweight multi-GPU
Edge Device Modeling						
nn-Meter [88]	Edge	M	Latency	<1%†	ms	Kernel detection
LitePred [18]	Edge	M	Latency	0.7%	ms	85-platform transfer
HELP [47]	Multi	M	Latency	1.9%	ms	10-sample adaptation
Compiler Cost Models						
TVM [12]	GPU	M	Schedule perf.	~15%	ms	Autotuning guidance
Ansor [89]	GPU	M	Schedule perf.	~15%	ms	Program sampling
TLP [87]	GPU	M	Tensor program	<10%	ms	Transformer cost model

coalescing, and L2 cache contention that progressively degrade analytical accuracy.

LLM-specific and compiler models. VIDUR [3] simulates LLM serving at <5% error; LIFE [19], HERMES [5], Omniwise [23], and SwizzlePerf [79] target inference. TVM [12]/Ansor [89] (~15% MAPE), TLP [87] (<10%), and SynPerf [81] target compiler autotuning [90].

5.3 Distributed Training and LLM Serving

Distributed systems require modeling communication, synchronization, and parallelism strategies [29, 64, 72]. ASTRA-sim [83] achieves 5–15% error via Chakra traces [73]; SimAI [80] reaches 1.9% at Alibaba scale; Echo [8] scales simulation to 10K+ devices; Lumos [51] 3.3% on H100s; PRISM [21] provides prediction intervals at 10K+ GPUs. Paleo [61] pioneered analytical estimation; MAD Max [28] and Sailor [74] extend it; Llama 3 [15] provides validation ground truth at 16K GPUs. The speed–fidelity hierarchy among these simulators reflects fundamentally different modeling granularities. VIDUR models serving at the *request level*: each prefill/decode phase is a single event with profiled duration, yielding second-scale simulation. ASTRA-sim operates at the *collective communication level*, replaying Chakra traces [73] to model compute-communication overlap critical for training. SimAI decomposes further to the *NCCL algorithm level*, modeling chunk-based ring/tree reductions—this matters because network congestion is non-linear: overlapping collectives that individually fit within bandwidth may congest, an effect invisible to per-collective models. SimAI's 1.9% MAPE (vs. ASTRA-sim's 5–15%) reflects this fidelity gain at production scale, though Echo [8] shows the cost: lightweight modeling is needed to scale to 10K+ devices.

For inference serving, VIDUR [3] models scheduling with vLLM [44]; DistServe [91] disaggregates prefill and decode for goodput optimization; Frontier [20] targets MoE; POD-Attention [24] and AQUA [71] address prefill-decode overlap and memory offloading respectively; ThrottLL'eM [36] models power effects; speculative decoding [9] creates a moving target for all simulators.

5.4 Edge Device Modeling

Hardware diversity makes per-device analytical modeling impractical. nn-Meter [88] claims <1% MAPE but is unverifiable due to dependency failures (Section 7); LitePred [18] achieves 0.7% across 85 platforms; HELP [47] reaches 1.9% with 10-sample meta-learning.

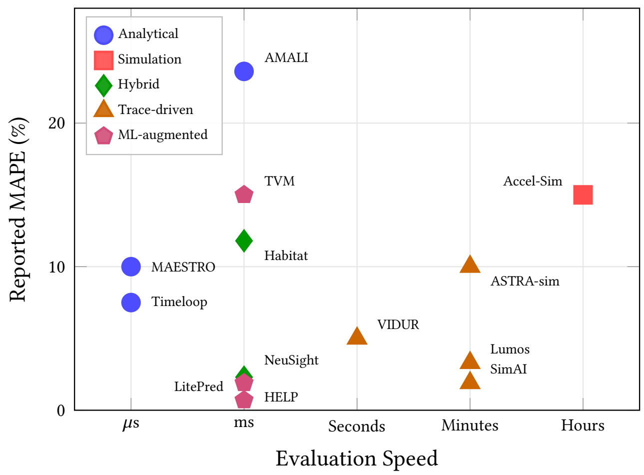

Figure 5: Self-reported accuracy vs. evaluation speed across surveyed tools. Each point represents a tool's MAPE on its *own* benchmarks and hardware—values are not directly comparable across tools targeting different platforms.

ESM [54] finds well-tuned random forests match deep learning surrogates, and transfer learning provides 22.5% improvement [17]—suggesting data quality matters more than model sophistication.

5.5 Cross-Cutting Themes

Three architectural insights emerge. *First*, structural decomposition aligned with hardware execution boundaries consistently outperforms black-box approaches: Timeloop's loop nests reflect systolic array dataflow, NeuSight's tiles mirror CUDA thread block scheduling, and VIDUR's prefill/decode split captures distinct compute- vs. memory-bound regimes. *Second*, the critical modeling features differ by platform: data reuse for accelerators, thread block occupancy for GPUs, and collective topology for distributed systems—explaining why no single methodology spans all platforms. *Third*, a persistent **accuracy–generality–speed trade-off** drives methodological diversity; subdomain maturity correlates with economic incentive: accelerator DSE is most mature (irreversible chip errors), distributed training is fastest-growing (million-dollar runs), and edge modeling has weakest reproducibility.

6 Comparison and Analysis

We analyze trade-offs across methodology types along accuracy and speed dimensions (see Table 3 for per-tool details); generalization and interpretability challenges are deferred to Section 8. Figure 5 visualizes the accuracy–speed trade-off space.

Caveat. The accuracy values in Figures 5 and 6 are *self-reported* on each tool's own benchmarks and hardware. No common evaluation benchmark exists for performance modeling tools, so these numbers are **not directly comparable across platforms or workload types**. We include them to illustrate *within-domain* trade-offs and identify difficult problem domains, not to rank tools against each other. A common-benchmark comparison is a key future direction (Section 8).

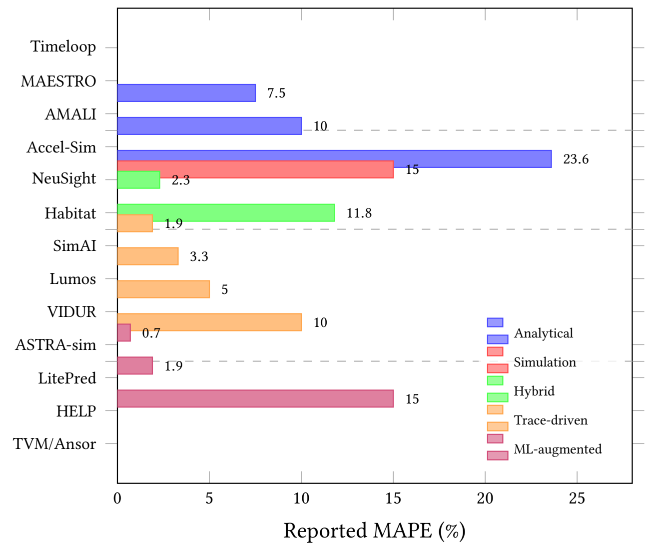

Figure 6: Self-reported accuracy (MAPE) of surveyed tools, grouped by methodology type. Range midpoints used where ranges are reported. Values are not directly comparable across tools: each was measured on different benchmarks, workloads, and hardware. Horizontal groupings by dashed lines separate distinct problem domains (accelerator, GPU, distributed, edge).

6.1 Accuracy by Problem Difficulty

We organize accuracy results by inherent problem difficulty rather than comparing across incompatible benchmarks (Figure 6). Accelerator modeling is most tractable (5–10%) due to deterministic data movement; GPU kernel prediction achieves 2–12% via hybrid methods; distributed systems reach 2–15% where communication modeling dominates error; edge prediction achieves 0.7–2% but requires per-device profiling. The architectural reasons for these difficulty tiers are analyzed in Sections 5.2 and 5.3.

6.2 Practitioner Tool Selection

Tool selection depends on target platform and acceptable error margin. Accelerator DSE: Timeloop/MAESTRO (μs-speed); Sparseloop for sparse workloads. GPU: NeuSight for accuracy–speed balance; Accel-Sim for μarch detail. Distributed: VIDUR for serving; ASTRA-sim/SimAI for training at scale. Edge: LitePred for coverage; HELP with minimal data. Additional factors include available hardware for profiling, team expertise with specific frameworks, integration with existing workflows, and license constraints. Prefer Docker-first tools (Section 7).

7 Experimental Evaluation

We conducted hands-on evaluations of five tools spanning methodology types: Timeloop (analytical), ASTRA-sim (trace-driven, distributed), VIDUR (trace-driven, LLM serving), nn-Meter (ML-augmented, edge), and NeuSight (hybrid, GPU). We selected one tool per methodology type to maximize coverage; we excluded proprietary tools

Table 4: VIDUR simulation results for Llama-2-7B inference serving on a simulated A100 GPU (Poisson arrivals at QPS 2.0, seed=42). All metrics from our own experiments.

Metric	vLLM	Sarathi
Requests	200	50
QPS (Poisson)	2.0	2.0
Avg E2E latency (s)	0.177	0.158
P99 E2E latency (s)	0.314	0.262
Avg TTFT (s)	0.027	0.025
Avg TPOT (s)	0.0093	0.0090
Requests preempted	0	0

(e.g., NVIDIA Nsight Compute, internal TPU profilers) as they cannot be independently reproduced.

Scope and limitations. All evaluations ran on Apple M2 Ultra (aarch64, 192 GB RAM) using Docker containers where provided. *No GPU hardware was available*, so we **do not validate accuracy claims**. Instead, we evaluate *reproducibility*: can a practitioner reproduce a tool's functionality without the original authors' environment? This complements accuracy evaluation, which would require common-benchmark runs on target hardware (Section 8). All three Docker-based tools (VIDUR, Timeloop, ASTRA-sim) reproduced successfully; NeuSight required manual setup but produced correct outputs; nn-Meter failed entirely.

7.1 Per-Tool Results

VIDUR. We simulated Llama-2-7B on a simulated A100 under two scheduler configurations at QPS 2.0 (Table 4). Sarathi [2] achieves lower latency than vLLM (avg 0.158 s vs. 0.177 s), consistent with its more efficient prefill–decode interleaving; neither scheduler triggered preemptions at this load level.

Timeloop. Docker CLI produces deterministic, bit-identical outputs for Eyeriss-like configurations; Python bindings fail (`ImportError: libbarvinok.so.23`).

ASTRA-sim. Collective microbenchmarks and ResNet-50 training at 2–8 simulated GPUs (Table 5) show internal consistency: Reduce-Scatter takes half the time of All-Reduce; communication overhead scales 5.76× for 4× more GPUs. Production-scale validation (100+ GPUs) would be needed to assess accuracy under realistic conditions.

NeuSight. Tile-based decomposition mirrors CUDA tiling for dense operations; irregular workloads had limited examples.

nn-Meter. After four attempts (>4h), no predictions ran: pickle-serialized predictors (scikit-learn 0.23.1) are incompatible with current versions.

7.2 Lessons and Threats to Validity

Key lessons: (1) Docker-first deployment correlates with reproducibility; ML model serialization is fragile (nn-Meter's pickle predictors became unusable within two years). (2) Reference outputs enable trust without hardware. (3) Scale-limited evaluation (2–8 GPUs) understates system tools [15].

Threats. Our venue-focused search may under-represent industry publications. We exclude proprietary tools (Nsight Compute [56], internal TPU models) from evaluation. Accuracy metrics vary across papers, limiting direct comparison—we caveat all cross-tool comparisons (Figures 5, 6). Our evaluation covers 5 of 22 tools, selected for methodology diversity; a complete study would include SimAI, AMALI, and Habitat.

Table 5: 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%

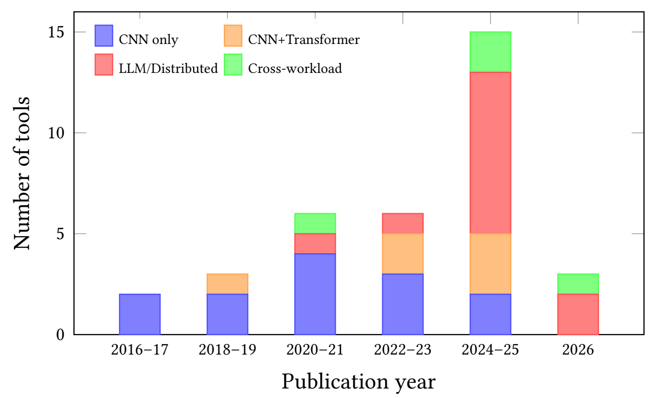

Figure 7: Workload coverage of surveyed tools by publication period. The shift toward transformer and LLM workloads accelerates from 2023, but MoE and diffusion models remain largely uncharacterized.

8 Open Challenges and Future Directions

Generalization gaps. *Workload*: The temporal validation lag (Section 4) is closing for transformers but remains wide for emerging workloads—MoE, diffusion [40], and dynamic inference lack validated tools; scaling laws [14, 22, 27, 37] predict loss but not latency. Figure 7 shows the shift toward LLM workloads since 2023. *Hardware*: cross-family transfer (GPU→TPU→PIM) remains unsolved despite meta-learning (HELP) and feature-based transfer (LitePred). *Temporal*: software stack evolution silently invalidating models is addressed by no tool.

The composition problem. Composing kernel-level predictions into end-to-end estimates is unsolved (Figure 8): kernel-level errors of 2–3% yield ∼10× higher variance at model level ($\sigma_{\text{model}} \approx$

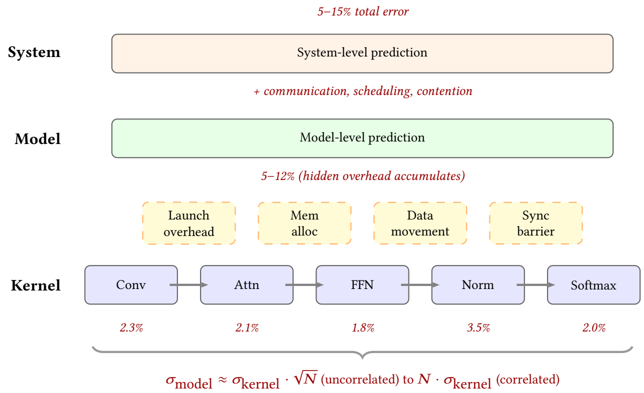

Figure 8: Error composition across abstraction levels. Kernel-level predictions (2–3% each) accumulate through hidden overheads (kernel launch, memory allocation, data movement, synchronization) that are not captured by kernel-level tools, yielding 5–12% model-level error. System-level errors add communication and scheduling overhead.

$\sigma_{\text{kernel}} \cdot \sqrt{N}$), and correlated errors can compound linearly. VIDUR sidesteps this by profiling entire prefill/decode phases.

Emerging hardware and future directions. PIM [26, 31, 46, 58], chiplets, and disaggregated designs blur memory hierarchy assumptions; FlashAttention [16] changes the landscape faster than models retrain; no MLPerf [53, 67] equivalent exists for performance *prediction*. Key future directions: (1) a common evaluation benchmark for modeling tools; (2) validated tools for frontier workloads; (3) formal composition error bounds; (4) unified energy-latency-memory prediction [66]; (5) Docker-first deployment with portable formats (ONNX, Chakra [73]).

9 Conclusion

This survey analyzed 22 tools for predicting ML workload performance. Key findings: no single methodology dominates—the right choice depends on practitioner priorities; LLM workloads demand specialized modeling (prefill/decode, KV cache, dynamic batching); Docker-first tools remain reproducible while serialized ML models become unusable; and accuracy claims require scrutiny due to varying benchmarks. The most pressing gaps are common evaluation benchmarks, validated tools for frontier workloads (MoE, diffusion), kernel-to-end-to-end error composition, and emerging hardware support.

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] Shirley Browne, Jack Dongarra, Nathan Garner, George Ho, and Philip Mucci. 2000. A Portable Programming Interface for Performance Evaluation on Modern Processors. *International Journal of High Performance Computing Applications* 14, 3 (2000), 189–204. https://doi.org/10.1177/109434200001400303 PAPI: portable API for hardware performance counters, foundational tool for performance analysis.

[8] 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).

[9] 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.

[10] 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.

[11] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 269–284. https://doi.org/10.1145/2541940.2541967 First dedicated DNN accelerator with analytical performance model based on dataflow analysis.

[12] 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.

[13] 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

[14] 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..

[15] 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..

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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..

[22] 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..

[23] 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.

[24] Yanbin Hao et al. 2025. POD-Attention: Unlocking Full Prefill-Decode Overlap for Faster LLM Inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1–15. Full overlap between prefill and decode phases for LLM inference.

[25] 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.

[26] 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..

[27] 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.

[28] 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

[29] 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.

[30] 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..

[31] 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..

[32] 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).

[33] Anand Jayarajan, Wei-Lin Hu, Gauri Zhao, and Gennady Pekhimenko. 2023. Sia: Heterogeneity-aware, Goodput-optimized ML-Cluster Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 642–657. https://doi.org/10.1145/3600006.3613175 Extends goodput optimization to heterogeneous GPU clusters for training workloads.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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

[39] 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.

[40] 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.

[41] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49. https://doi.org/10.1109/LCA.2015.2414456 Fast extensible DRAM simulator supporting DDRx, LPDDRx, GDDRx, WIOx, HBMx standards.

[42] 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

[43] 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

[44] 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

[45] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308 Multi-level IR infrastructure enabling cost model composition across abstraction levels.

[46] 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..

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] NVIDIA Corporation. 2019. Nsight Compute: Interactive Kernel Profiler. https://developer.nvidia.com/nsight-compute. Industry-standard GPU kernel profiling tool with roofline analysis.

[57] 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

[58] 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..

[59] 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.

[60] 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.

[61] 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

[62] Aurick Qiao, Sang Keun Agrawal, Anand Jayarajan, Moustafa Mittal, Amar Altaf, Michael Cho, and Gennady Pekhimenko. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 1–18. Goodput estimation for co-optimizing resource allocation and training hyperparameters.

[63] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 519–530. https://doi.org/10.1145/2491956.2462176 Pioneered separation of algorithm and schedule with learned cost models for autoscheduling.

[64] 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.

[65] 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

[66] 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.

[67] 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.

[68] Arun F. Rodrigues, K. Scott Hemmert, Brian W. Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, R. Risen, Jeanine Cook, Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2012. The Structural Simulation Toolkit. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38. 37–42. https://doi.org/10.1145/1964218.1964225 Modular framework for system-level simulation, widely used for HPC and interconnect modeling.

[69] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters* 10, 1 (2011), 16–19. https://doi.org/10.1109/L-CA.2011.4 Widely-used cycle-accurate DDR2/DDR3 memory simulator validated against manufacturer Verilog models.

[70] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2019. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 58–68. https://doi.org/10.1109/ISPASS.2019.00016 Cycle-accurate systolic array simulator for DNN accelerator DSE.

[71] Zhuomin Shen, Jaeho Kim, et al. 2025. AQUA: Network-Accelerated Memory Offloading for LLMs in Scale-Up GPU Domains. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 1–16. https://doi.org/10.1145/3676641.3715983 Improves LLM inference responsiveness by 20x through network-accelerated memory offloading.

[72] 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.

[73] 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).

[74] 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..

[75] 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

[76] 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.

[77] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*. 10–19. https://doi.org/10.1145/3315508.3329973 Tile-based GPU programming with heuristic performance model for kernel generation.

[78] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW)*. 207–216. https://doi.org/10.1109/ICPPW.2010.38 Lightweight tools for thread/cache topology, affinity, and performance counter measurement.

[79] 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.

[80] 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..

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

[82] 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

[83] 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

[84] 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

[85] 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.

[86] 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.

[87] 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

[88] 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.

[89] 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.

[90] 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.

[91] 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.