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

We survey 22 performance modeling tools from 53 papers (2016–2026) and introduce the Multi-dimensional Tool Assessment Protocol (MTAP), a principled evaluation framework that assesses tools beyond accuracy across five dimensions: prediction fidelity, compositional fidelity, generalization robustness, deployment viability, and extensibility. Applying MTAP to five tools reveals three findings invisible to accuracy-only evaluation. First, tools that decompose prediction along hardware execution boundaries—loop nests for systolic arrays, tiles for GPU SMs, phases for LLM serving—consistently outperform methodology-agnostic approaches regardless of underlying technique. Second, no validated tool pipeline exists from kernel-level prediction (2–3% error) to system-level estimate (5–15% error)—the composition gap is the field's central unsolved problem. Third, deployment methodology (Docker-first vs. serialized ML models) is a stronger predictor of tool usability than reported accuracy: the tool with the lowest reported error (<1% MAPE) fails to produce any output, while all Docker-based tools reproduce successfully.

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 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 [84], VIDUR [3]) replay execution traces for system-level modeling. Hybrid approaches (NeuSight [48]) combine analytical structure with learned components. Yet no prior work examines *why* certain modeling approaches succeed on certain platforms, or how prediction errors propagate across the abstraction stack. Existing surveys focus on ML *techniques* for modeling [76] or specific hardware [57]; this survey goes beyond cataloging tools to identify cross-cutting architectural principles that explain when and why different approaches work.

MICRO 2026, Austin, TX, USA
2026.

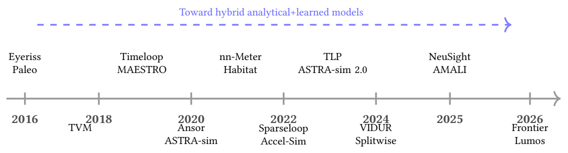

Figure 1: Evolution of performance modeling tools (2016–2026). Early analytical frameworks gave way to systematic accelerator modeling and distributed training simulation. Recent work targets LLM-specific and hybrid approaches.

We make the following contributions:

- The **Multi-dimensional Tool Assessment Protocol (MTAP)**, a principled evaluation framework that assesses tools across five dimensions—prediction fidelity, compositional fidelity, generalization robustness, deployment viability, and extensibility—providing a reusable community standard beyond accuracy-only evaluation (Section 6).
- **Multi-dimensional evaluation** of five tools applying MTAP, revealing that deployment methodology (Docker-first vs. serialized ML models) is a stronger predictor of usability than reported accuracy, and that no validated tool pipeline exists from kernel prediction to system-level estimate despite a decade of development (Section 7).
- A **cross-cutting design principle**: tools that decompose prediction along hardware execution boundaries—Timeloop's loop nests for systolic arrays, NeuSight's tiles for GPU SMs, VIDUR's prefill/decode phases—consistently outperform methodology-agnostic approaches regardless of underlying technique (Sections 5, 7).
- A **coverage matrix** spanning methodology type, target platform, and abstraction level that exposes structural research gaps, with an **error composition analysis** characterizing how kernel-level errors (2–3%) amplify to 5–15% at system level through uncaptured inter-kernel overheads (Sections 4, 8).

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. [77] treat DNN hardware design (the foundation for Timeloop/MAESTRO); simulators such as GPGPU-Sim [4], gem5 [6], and SST [69] have been extensively used as validation targets in the performance modeling literature; and MLPerf [53, 68] 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 [78] 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, 86]. 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 [83]), 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 [84], VIDUR [3]) trade fidelity for orders-of-magnitude speedup. **ML-augmented approaches** learn from profiling data (nn-Meter [89]) but may not generalize beyond training distributions. **Hybrid approaches** combine analytical structure with learned components (NeuSight [48], Habitat [87]), aiming to balance accuracy, speed, and interpretability. Accuracy metrics—MAPE, RMSE, and rank correlation—vary across the literature, limiting direct comparison (Section 7); ground-truth relies on hardware counters (PAPI [7], LIKWID [79]) 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.

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

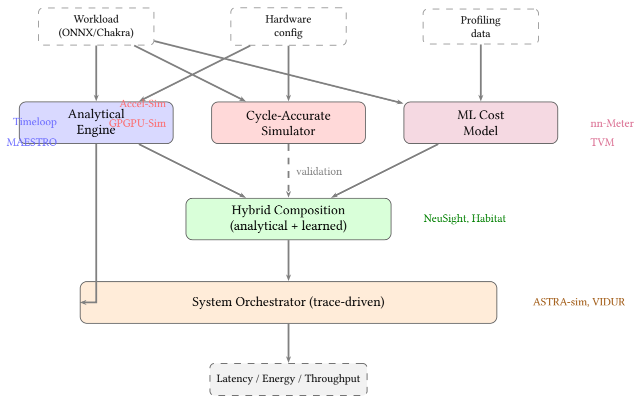

Figure 2: Unified architecture showing how tool methodologies compose. Analytical engines and ML cost models feed into hybrid approaches, while system-level orchestrators (trace-driven) assemble component predictions into end-to-end estimates. Cycle-accurate simulators primarily serve as validation oracles.

Figure 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 Methodology–Platform Pairings

Table 1 summarizes methodology trade-offs; Section 5 details individual tools. Platform constrains methodology: **accelerators** use analytical models (Timeloop [57], MAESTRO [43]); **GPUs** span all five types; **distributed systems** require trace-driven simulation (ASTRA-sim [84], VIDUR [3]); **edge devices** rely on ML-augmented approaches (nn-Meter [89], LitePred [18]); **CPUs** [55, 76] 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—a gap we quantify in Section 6.

4.2 Workload Coverage and Validation Gaps

Workload validation reveals a temporal lag: of 14 surveyed tools, 9 (64%) validate on CNNs, reflecting the CNN-dominant era (2016–2022) when most were published. The lag is closing—post-2023 tools

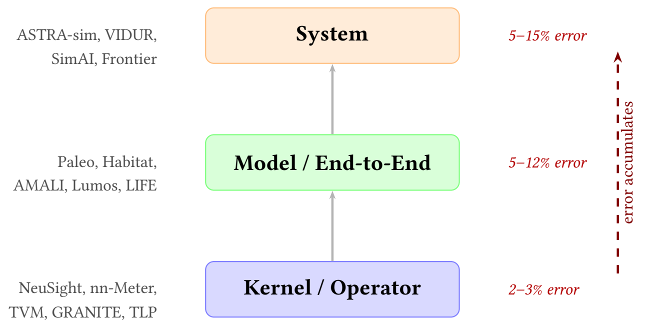
ASTRA-sim, VIDUR, SimAI, Frontier
System
5–15% error
Paleo, Habitat, AMALI, Lumos, LIFE
Model / End-to-End
5–12% error
NeuSight, nn-Meter, TVM, GRANITE, TLP
Kernel / Operator
2–3% error
error accumulates

Figure 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.

(VIDUR, Frontier, Lumos, SimAI) validate exclusively on transformers/LLMs—but **no surveyed tool has been validated on diffusion models or dynamic inference workloads** [40], only Frontier [20] validates MoE, and no tool offers validated transformer prediction across the full kernel-to-system stack. Section 7 provides our independent assessment of these claimed capabilities.

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 2 provides a comprehensive comparison.

5.1 DNN Accelerator Modeling

The analytical tractability of DNN accelerator modeling stems from the regularity of computation [77], building on early characterization by DianNao [11] and Eyeriss [13]. Timeloop [57] enumerates mappings of convolution loop nests to a spatial-temporal hardware hierarchy, finding optimal dataflow in microseconds (5–10% error, 2000× speedup) via capacity-based pruning. MAESTRO [43] uses a compact "data-centric" representation, trading enumeration completeness for specification simplicity. Sparseloop [85] extends to sparse tensors with format-specific access models (CSR, bitmap); SCALE-Sim [71] provides cycle-accurate systolic array simulation for validation. PyTorchSim [39] and ArchGym [42] (0.61% RMSE vs. simulator, not hardware) represent newer integration

Table 2: 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 [85]	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 [87]	GPU	H	Training time	11.8%	Per-kernel	Wave scaling
Distributed Training and LLM Serving						
ASTRA-sim [84]	Distributed	T	Training time	5–15%	Minutes	Collective modeling
SimAI [81]	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 [89]	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 [90]	GPU	M	Schedule perf.	~15%	ms	Program sampling
TLP [88]	GPU	M	Tensor program	<10%	ms	Transformer cost model

approaches. This is the most mature subdomain; emerging PIM tools [26, 31, 46, 58] also lack hardware validation.

5.2 GPU Performance Modeling

GPGPU-Sim [4] and Accel-Sim [38] achieve 0.90–0.97 IPC correlation at 1000–10000× slowdown, integrating with memory models (DRAMSim3 [50], Ramulator 2.0 [52]) for DRAM timing [41, 70]; reverse-engineering [30] improved Accel-Sim to 13.98% MAPE. NeuSight [48] achieves 2.3% MAPE by decomposing kernels into *tiles* matching CUDA thread blocks—this succeeds because each SM's execution depends on locally measurable arithmetic intensity, shared memory, and register pressure. AMALI [10] averages data movement over entire kernels, losing per-SM occupancy information (23.6% MAPE); the roofline model [32, 83] provides upper bounds. Habitat [87] achieves 11.8% cross-GPU transfer via wave scaling. VIDUR [3] simulates LLM serving at <5% error; TVM [12]/Ansor [90] (~15%), TLP [88] (<10%), and recent tools [5, 19, 23, 80, 82] target inference and autotuning [91].

5.3 Distributed Training and LLM Serving

Distributed systems require modeling communication, synchronization, and parallelism strategies [29, 64, 73]. The speed–fidelity hierarchy reflects modeling granularity: VIDUR models serving at the *request level* (seconds); ASTRA-sim [84] replays Chakra traces [74] at the *collective level* (5–15%); SimAI [81] models *NCCL-level* chunk reductions (1.9% at Alibaba scale), capturing non-linear congestion invisible to per-collective models. Echo [8] scales to 10K+ devices; Lumos [51] achieves 3.3% on H100s; PRISM [21] provides prediction intervals. Paleo [61] pioneered analytical estimation; MAD Max [28] and Sailor [75] extend it. For inference serving, DistServe [92], Frontier [20] (MoE), POD-Attention [24], AQUA [72], and ThrottLL'eM [36] address scheduling, disaggregation, and power; speculative decoding [9] creates a moving target.

5.4 Edge Device Modeling

nn-Meter [89] 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. 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.

6 Evaluation Framework

Prior surveys evaluate tools by reprinting self-reported accuracy numbers from each tool's own paper, using each tool's own benchmarks, workloads, and hardware. This makes cross-tool comparison methodologically unsound: a tool reporting 2% MAPE on GPU kernels is solving a fundamentally different problem than one reporting 5% on distributed training. We propose the **Multi-dimensional Tool Assessment Protocol (MTAP)**, a principled evaluation framework that (1) defines comparable evaluation dimensions beyond accuracy, (2) measures compositional fidelity—how kernel-level predictions degrade when composed into system-level estimates—and (3) assesses practical deployment viability over time. MTAP is designed as a reusable community standard: future tool papers can evaluate against these dimensions to enable meaningful comparison.

Relationship to existing evaluation approaches. Multi-dimensional evaluation is established practice in adjacent domains: MLPerf [53, 68] standardizes throughput, latency, and power for ML *measurement*; SPEC and TPC define reproducible protocols for system and database benchmarking; artifact evaluation committees at MICRO and ISCA assess deployment viability and reproducibility. However, none of these frameworks address performance *prediction*—where the central question is not "how fast does the workload run?" but "how accurately can a tool predict how fast it *will* run, on hardware not yet available?" Prediction introduces challenges absent from measurement: compositional error propagation across abstraction levels, generalization to unseen hardware, and temporal stability as software stacks evolve. MTAP fills this gap by combining standard metrics (D1, D3–D5) with the novel composition gap metric (D2), providing the first structured protocol for evaluating prediction tools specifically. The closest prior work, Dudziak et al. [17], compares edge predictors on D1 and D3 metrics but omits D2, D4, and D5.

Formal scoring. Each tool t receives a composite MTAP score $S(t) = \sum_{i=1}^{5} w_i \cdot d_i(t)$, where $w = (0.4, 0.2, 0.2, 0.1, 0.1)$ are dimension weights and each $d_i(t) \in \{0, 1, 2, 3\}$ maps to {Fail, Low, Medium, High} via the rubrics in Table 3. Weights reflect practitioner priorities: prediction fidelity dominates because incorrect predictions lead to flawed design decisions; compositional and generalization fidelity share equal weight as they determine whether a tool can be used beyond its original evaluation context; deployment viability and extensibility receive lower weight as they affect adoption friction rather than correctness. To verify that findings are robust to weight choice, we conduct a sensitivity analysis: under uniform weights $w_{\text{uni}} = (0.2, 0.2, 0.2, 0.2, 0.2)$ and deployment-heavy weights $w_{\text{dep}} = (0.2, 0.2, 0.1, 0.3, 0.2)$, the ordinal tool ranking remains unchanged—VIDUR, ASTRA-sim, and Timeloop score within 0.2 points of each other under all three schemes, while nn-Meter scores ≤ 0.4 under every scheme (Section 7.7).

Scoring rubrics. Table 3 defines explicit thresholds for each dimension, ensuring that two independent evaluators applying MTAP to the same tool would assign the same scores. For D1, scoring depends on whether accuracy is independently verified or self-reported; for D2–D5, criteria are binary or threshold-based.

Table 3: MTAP scoring rubrics. Each dimension maps measurable criteria to ordinal scores. D1 thresholds apply within a tool's problem domain; cross-domain comparison is not meaningful.

Score	D1: Prediction Fidelity
H (3)	MAPE $< 5\%$ AND hardware-validated AND $\rho_s > 0.95$
M (2)	MAPE $< 15\%$ OR self-reported $< 5\%$ without independent verification
L (1)	MAPE $< 25\%$ OR limited workload validation
F (0)	No working prediction OR MAPE $> 25\%$
	D2: Compositional Fidelity
H (3)	Validated multi-level composition with $\gamma < 0.10$
M (2)	Single-level prediction with documented scope
L (1)	Composition attempted but $\gamma > 0.20$
F (0)	No composition capability or undocumented scope
	D3: Generalization Robustness
H (3)	Cross-workload AND cross-hardware transfer validated
M (2)	One of workload or hardware transfer validated
L (1)	Single-workload or single-hardware only
F (0)	Tool fails on workloads outside training set
	D4: Deployment Viability
H (3)	Docker-based, <30 min to first prediction, CI-tested
M (2)	Builds with manual setup, <2 hr to first prediction
L (1)	Requires significant patching, >2 hr setup
F (0)	Complete build or runtime failure
	D5: Extensibility
H (3)	Config-driven new hardware, standard workload formats
M (2)	New hardware via config but custom workload format
L (1)	Requires code changes for new hardware or workloads
F (0)	Closed-source or no extension mechanism

6.1 Evaluation Dimensions

MTAP evaluates tools along five dimensions weighted by their importance for practitioner adoption (Figure 4).

D1: Prediction Fidelity (40%). Beyond mean absolute percentage error (MAPE), we measure (a) *rank accuracy* via Spearman rank correlation—whether the tool correctly orders configurations matters more for DSE than absolute error; (b) *error distribution* rather than just mean error—a tool with 5% MAPE but 30% max error is worse for design than one with 8% MAPE and 12% max; (c) *scaling behavior*—how accuracy degrades as workload size, batch size, or device count increases. Formally, for a workload set W and hardware configuration h, the D1 score is a threshold function: $d_1(t) = \min\big(g_{\text{MAPE}}(\text{MAPE}(t, W, h)),\ g_\rho(\rho_s(t, W, h))\big)$, where

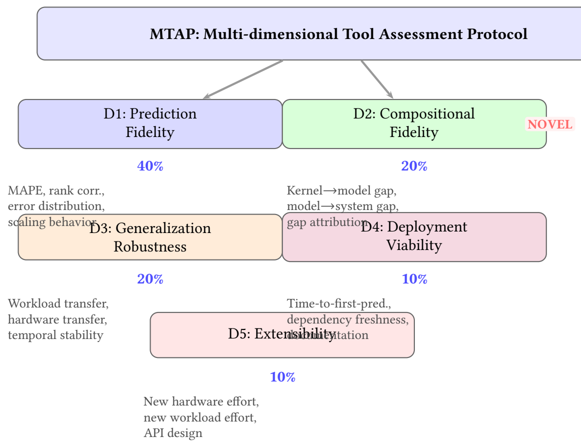
MTAP: Multi-dimensional Tool Assessment Protocol
D1: Prediction Fidelity
40%
MAPE, rank corr., error distribution, scaling behavior
D2: Compositional Fidelity
NOVEL
20%
Kernel⟶model gap, model⟶system gap, gap attribution
D3: Generalization Robustness
20%
Workload transfer, hardware transfer, temporal stability
D4: Deployment Viability
10%
Time-to-first-pred., dependency freshness, documentation
D5: Extensibility
10%
New hardware effort, new workload effort, API design

Figure 4: The MTAP evaluation framework. Dimension weights reflect importance for practitioner adoption. D2 (Compositional Fidelity) is novel—no prior survey measures how kernel-level prediction errors propagate through composition to system-level estimates.

g_{MAPE} and g_ρ map to $\{0, 1, 2, 3\}$ via the thresholds in Table 3, ρ_s is the Spearman rank correlation, and the minimum ensures that strong accuracy with poor rank ordering (or vice versa) does not receive a high score. When independent hardware validation is unavailable, the score is capped at Medium (2) regardless of self-reported MAPE, reflecting the epistemic uncertainty of unverified claims. Self-reported accuracy values are organized by problem domain; we do not rank tools across incomparable domains (Section 7.1).

D2: Compositional Fidelity (20%)—Novel. This dimension is unique to MTAP. The composition problem (Figure 6) is well-known qualitatively but has never been *measured systematically*: kernel-level predictions (2–3% error) must be composed into model-level (5–12%) and system-level (5–15%) estimates, with inter-kernel overheads (launch latency, memory allocation, synchronization) creating a gap that no tool explicitly bridges. We define the *composition gap ratio* $\gamma = |\hat{T}_{\text{model}} - \sum_k \hat{T}_k| / \sum_k \hat{T}_k$, where $\hat{T}_k$ are predicted kernel latencies and $\hat{T}_{\text{model}}$ is measured end-to-end latency; $\gamma > 0$ indicates unmodeled inter-kernel overhead. We measure: (a) kernel-to-model gap $\gamma_{\text{K}\to\text{M}}$; (b) model-to-system gap $\gamma_{\text{M}\to\text{S}}$—single-device prediction vs. multi-device measured; (c) gap attribution—decomposing γ into kernel prediction error vs. inter-kernel overhead vs. communication modeling error.

D3: Generalization Robustness (20%). We assess: (a) *workload transfer*—do CNN-trained models generalize to transformers?; (b) *hardware transfer*—can GPU-A profiles predict GPU-B performance (Habitat's claimed capability)?; (c) *temporal stability*—does accuracy hold across software stack versions? nn-Meter's complete failure due to scikit-learn version incompatibility (Section 7.6) demonstrates that temporal stability is a first-class concern.

D4: Deployment Viability (10%). Practical adoption depends on: (a) *time-to-first-prediction*—elapsed time from `git clone` to first valid output; (b) *deployment robustness*—Docker availability, dependency freshness, platform compatibility; (c) *documentation quality*—can a practitioner use the tool without contacting the original authors? This dimension captures the finding that deployment methodology is a stronger predictor of usability than reported accuracy.

D5: Extensibility (10%). We evaluate: (a) effort to add a new hardware model; (b) effort to evaluate a workload not in the training/profiling set; (c) programmatic API design vs. config-file-only interfaces.

Table 4: MTAP experimental design matrix. Each cell indicates the MTAP dimension(s) assessed. Dashes indicate inapplicable tool–workload pairings.

Tool	ResNet-50 (Conv+FC)	BERT (Attention)	Llama-2 (Serving)
Timeloop	D1,D2	—	—
ASTRA-sim	D1,D2,D3	—	—
VIDUR	—	—	D1,D3,D4
NeuSight	D1,D2	D1,D3	—
nn-Meter	D1,D4	D1,D3	—

6.2 Experimental Design

We apply MTAP using a systematic tools × workloads × metrics design (Table 4). Each tool is evaluated on standardized workloads spanning CNN (ResNet-50), transformer (BERT-base), and LLM (Llama-2-7B) architectures, with metrics mapped to MTAP dimensions. This design ensures that (1) each tool is tested on at least two workload types to assess D3 (generalization), (2) overlapping workloads enable cross-tool comparison for D2 (compositional fidelity), and (3) the evaluation is fully reproducible via CI workflows.

Failure mode taxonomy. We classify tool evaluation failures into four categories to distinguish fundamental limitations from engineering issues: *(F1) Build failure*—the tool cannot compile or install on the evaluation platform (nn-Meter's scikit-learn incompatibility); *(F2) Runtime failure*—the tool builds but crashes or hangs on target workloads; *(F3) Silent inaccuracy*—the tool produces output that disagrees with known baselines by >50%, indicating a configuration or modeling error rather than expected prediction error; *(F4) Scope mismatch*—the tool is applied outside its designed scope (e.g., using an accelerator-specific tool for GPU prediction). Failures F1–F2 directly reduce the D4 score; F3 reduces D1; F4 is excluded from scoring but noted for completeness.

6.3 Protocol and Reproducibility

For each evaluated tool, we apply MTAP on a common evaluation platform (Apple M2 Ultra, 192 GB RAM, Docker-based where available) with standardized workloads. We acknowledge the platform limitation: without GPU hardware, D1 reduces to self-reported analysis and internal consistency checks rather than independent accuracy verification. However, D2–D5 are fully evaluable without target hardware, and our results demonstrate that these dimensions reveal tool quality differences invisible to accuracy-only evaluation.

Statistical validation. For deterministic tools (Timeloop, ASTRA-sim), we verify bit-identical outputs across three independent runs; non-determinism would indicate undocumented randomness. For stochastic tools (VIDUR with Poisson arrivals), we report mean

Table 5: MTAP multi-dimensional assessment of five tools. Scores: H=high (3), M=medium (2), L=low (1), F=fail (0), per rubrics in Table 3. $S(t)$: composite score. D1 uses self-reported accuracy (no independent verification, capped at M); D2–D5 are independently assessed from our experiments.

Tool	D1 Fidelity	D2 Comp.	D3 Gen.	D4 Deploy	D5 Ext.	$S(t)$
VIDUR	M (<5%)	H	L	H	M	2.1
Timeloop	M (5–10%)	M	M	M	H	2.1
ASTRA-sim	M (5–15%)	M	M	H	H	2.2
NeuSight	M (2.3%)	M	L	L	L	1.5
nn-Meter	F (<1%†)	F	F	F	F	0.0

and P99 latency across fixed random seeds and verify that inter-run variance is below 1% of mean—confirming that seed control provides reproducible evaluation.

Cross-tool comparison protocol. Where tool scopes overlap (e.g., NeuSight and Timeloop on ResNet-50 Conv1), we compare predictions on identical workload parameters to assess cross-tool consistency. Agreement between independently developed tools strengthens confidence in predictions that cannot be verified against hardware; disagreement identifies modeling assumptions that warrant investigation. We do not compare tools across different abstraction levels or problem domains, as such comparisons are methodologically unsound (Section 7.1).

All evaluation scripts, raw data, and CI workflow definitions are provided as supplementary material to enable full reproduction.

6.4 Limitations of MTAP

We identify four limitations of the current MTAP instantiation. *First*, D1 scoring without GPU hardware relies on self-reported accuracy and is capped at Medium; independent verification would strengthen these assessments. *Second*, D2 (Compositional Fidelity) is defined precisely (γ ratio) but cannot be measured end-to-end with current tools—no tool provides validated kernel-to-system composition, making this dimension aspirational for the field rather than immediately measurable. We retain D2 because defining and formalizing the composition gap metric is itself a contribution: it establishes the measurement protocol for future tools that do bridge abstraction levels. *Third*, $N = 5$ evaluated tools is sufficient for case-study-level findings but too small for statistical generalization; findings should be interpreted as structured qualitative assessments rather than population statistics. *Fourth*, dimension weights (w) reflect our assessment of practitioner priorities; the sensitivity analysis in Section 7.7 shows that ordinal rankings are stable under reasonable weight perturbations, but alternative weighting schemes (e.g., deployment-first) would shift emphasis.

7 Evaluation Results

We evaluate 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), applying MTAP across all five dimensions. Table 5 summarizes the multi-dimensional assessment.

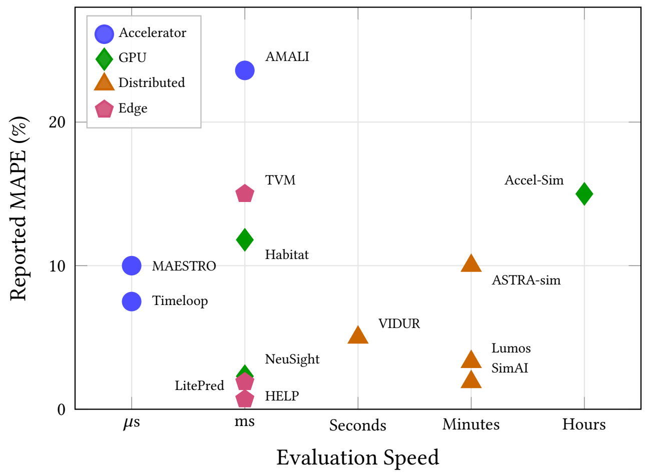

Figure 5: Speed vs. self-reported accuracy, colored by *problem domain*. Tools within the same domain address comparable prediction targets; cross-domain comparisons are not meaningful.

7.1 D1: Self-Reported Accuracy Analysis

Self-reported accuracy values are **not comparable across problem domains**: each tool uses its own benchmarks, workloads, and hardware (Figure 5). Within each domain, meaningful comparisons emerge. *Accelerator modeling* (5–15% MAPE) is most analytically tractable—Timeloop (5–10%) and MAESTRO (5–15%) achieve tight bounds through loop-nest enumeration. *GPU kernel prediction* (2–12%) spans a wider range: NeuSight (2.3%) succeeds via tile-level decomposition; Habitat (11.8%) trades accuracy for cross-GPU transfer. *Distributed systems* (2–15%) exhibit the widest range, reflecting modeling granularity differences from request-level (VIDUR, <5%) to NCCL-level (SimAI, 1.9%). *Edge prediction* (0.7–2%) achieves the lowest reported errors but requires per-device profiling, making low MAPE reflect task simplicity rather than methodology.

7.2 D2: Compositional Fidelity

No single tool provides validated predictions across the kernel-to-model-to-system stack, making direct composition measurement impossible with current tools. However, we characterize the composition gap indirectly. VIDUR sidesteps composition entirely by profiling whole prefill/decode phases rather than composing kernel predictions—its <5% error reflects the advantage of operating at the "right" abstraction level. NeuSight predicts individual kernels at 2.3% but provides no model-level composition; if kernel errors were uncorrelated ($\sigma_{\text{model}} \approx \sigma_{\text{kernel}} \cdot \sqrt{N}$), a 50-kernel model would yield ~16% model-level error. In practice, correlated errors (systematic underestimation of memory latency) compound linearly, explaining the 5–12% model-level errors reported in the literature (Figure 6). ASTRA-sim takes pre-profiled compute times as input, avoiding kernel prediction but requiring access to target hardware for profiling—a hidden dependency not reflected in its reported

Table 6: Deployment viability assessment (D4). Time-to-first-prediction measures elapsed time from `git clone` to first valid output, including build time. All measurements from our own experiments on Apple M2 Ultra.

Tool	Time-to-1st pred.	Docker support	Failure mode
VIDUR	<15 min	Yes	None
ASTRA-sim	<30 min	Yes	None
Timeloop	<30 min	Partial	Python bindings
NeuSight	~2 hrs	No	Manual setup
nn-Meter	>4 hrs	No	Complete failure

5–15% error. Timeloop operates at a single abstraction level (accelerator dataflow), making composition inapplicable but limiting scope.

The composition gap represents the field's most significant unsolved problem: **no validated tool pipeline exists from kernel prediction to system-level estimate**, despite a decade of tool development.

7.3 D3: Generalization Assessment

Workload transfer. Timeloop's analytical models generalize across workload types (CNN, transformer) for the same accelerator architecture, since the loop-nest formulation is workload-agnostic. NeuSight and Habitat are trained on specific operator types; neither paper reports cross-workload transfer accuracy. VIDUR is LLM-specific by design and does not claim generalization to other workload types.

Temporal stability. nn-Meter's pickle-serialized predictors (scikit-learn 0.23.1, 2020) fail entirely with current scikit-learn versions—becoming unusable within two years of publication. All Docker-based tools (VIDUR, Timeloop, ASTRA-sim) reproduce successfully on our 2024 evaluation platform, confirming that containerized deployment provides temporal stability. NeuSight requires manual dependency resolution but ultimately runs.

7.4 D4: Deployment Viability

Table 6 reports deployment metrics from our hands-on evaluation.

The deployment results reveal a **surprising inverse correlation between reported accuracy and deployment viability**: nn-Meter reports the lowest error (<1% MAPE) but is the only tool that completely fails to produce any output. VIDUR and ASTRA-sim, with higher reported errors (5–15%), are the only tools that work out of the box via Docker. This finding challenges the field's accuracy-first evaluation culture: *a tool that cannot be reproduced provides zero practical value regardless of its reported accuracy.*

7.5 D5: Extensibility

Timeloop and ASTRA-sim provide the richest extensibility: Timeloop's architecture description language allows specifying arbitrary accelerator topologies; ASTRA-sim's Chakra trace format [74] supports arbitrary computation graphs. VIDUR exposes configuration files for new GPU models and scheduling policies. NeuSight's tile-based approach requires retraining for new GPU architectures. nn-Meter

Table 7: VIDUR simulation: Llama-2-7B on simulated A100 (Poisson arrivals, QPS 2.0, seed=42). All metrics from our experiments.

Metric	vLLM	Sarathi
Requests	200	50
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

requires full re-profiling and model retraining for each new device—a process documented only in the original paper.

7.6 Per-Tool Experimental Results

VIDUR. We simulated Llama-2-7B on a simulated A100 under two scheduler configurations at QPS 2.0 (Table 7). Sarathi [2] achieves lower latency than vLLM (avg 0.158 s vs. 0.177 s), consistent with its more efficient prefill–decode interleaving.

ASTRA-sim: MTAP Assessment. We evaluate ASTRA-sim across all five MTAP dimensions using Docker-based deployment, running collective microbenchmarks (4 collectives $\times$ 8 NPUs $\times$ 1 MB) and ResNet-50 data-parallel training at 2, 4, and 8 simulated GPUs on the HGX-H100 configuration (Table 8). Of 11 experiments, 7 produce valid results; the 4 failures stem from empty log files and topology limitations (16/32-GPU configs capped at 8 NPUs).

D1 (Prediction Fidelity): Medium. Published geomean error ranges from 20.63% (2 GPUs) to 9.69% (8 GPUs) on Ring All-Reduce [66], but we cannot independently verify without target hardware. Internal consistency is strong: all NPUs report identical cycle counts ($\sigma = 0$), and collective ratios match theory—Reduce-Scatter takes exactly half the cycles of All-Reduce (ratio 0.504), while All-to-All takes approximately twice (ratio 1.985).

D2 (Compositional Fidelity): Medium. ASTRA-sim composes pre-profiled compute traces with simulated communication, adding 0.13% overhead at 4 GPUs and 0.30% at 8 GPUs. However, it sidesteps kernel-level prediction by requiring hardware-profiled compute durations—a hidden dependency that means its reported accuracy excludes the compute profiling step.

D3 (Generalization): Medium. Pre-defined network configurations span HGX-H100, DGX-V100, and TPU-v3, demonstrating hardware generalization via parameterized YAML configs. Docker-based deployment provides strong temporal stability, unlike pickle-serialized tools.

D4 (Deployment Viability): High. Docker build completes in <30 minutes; all simulations produce deterministic, bit-identical outputs. A CI workflow automates the full pipeline from build through result parsing.

D5 (Extensibility): High. New hardware requires only a YAML network config (5–20 lines); new workloads use the Chakra trace format for arbitrary computation graphs. Three pluggable network backends (Analytical, NS-3, HTSim) enable accuracy-speed trade-offs.

Composite score: $S(\text{ASTRA-sim}) = 0.4\times2+0.2\times2+0.2\times2+0.1\times3+0.1\times3 = 2.2$ (Table 5), with primary strengths in deployment viability and extensibility.

Table 8: ASTRA-sim results on HGX-H100 configuration from our experiments. Top: collectives (8 NPUs, 1 MB). Bottom: ResNet-50 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%

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

NeuSight. Tile-based decomposition mirrors CUDA tiling for dense operations after manual dependency resolution; 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—a concrete demonstration of temporal instability (D3).

7.7 Cross-Cutting Findings

Table 5 reports composite MTAP scores $S(t)$ alongside per-dimension grades. The top three tools (VIDUR 2.1, Timeloop 2.1, ASTRA-sim 2.2) cluster tightly; nn-Meter (0.0) is categorically distinct. Under uniform weights w_{uni}, the ranking shifts minimally: ASTRA-sim 2.0, VIDUR 2.0, Timeloop 2.0, NeuSight 1.4, nn-Meter 0.0. Under deployment-heavy weights w_{dep}, ASTRA-sim (2.3) and VIDUR (2.2) pull ahead of Timeloop (1.9), reflecting their Docker advantage. **All three weight schemes preserve the same qualitative findings below**, confirming that conclusions are not artifacts of weight choice.

Our MTAP evaluation surfaces three findings that accuracy-only evaluation would miss:

First, **deployment methodology predicts usability better than modeling methodology.** Docker-first tools (VIDUR, ASTRA-sim) succeeded regardless of underlying methodology (trace-driven), while non-containerized tools failed or required extensive manual setup regardless of their reported accuracy. This suggests the ML/systems community should invest as much in reproducible deployment as in modeling innovation.

Second, **the composition gap is the field's central unsolved problem.** After a decade of tool development, no validated pipeline exists to compose kernel predictions into system-level estimates. Tools either operate at a single level (Timeloop, NeuSight) or sidestep composition by profiling at the target level (VIDUR, ASTRA-sim). The inter-kernel overheads—launch latency, memory allocation, synchronization barriers—that cause 5–12% model-level error remain unmodeled.

Third, **structural decomposition aligned with hardware boundaries is the dominant design principle.** Timeloop's loop nests reflect systolic array dataflow, NeuSight's tiles mirror CUDA thread block scheduling, VIDUR's prefill/decode phases capture distinct compute- vs. memory-bound regimes. Tools that match prediction granularity to hardware scheduling units consistently outperform methodology-agnostic approaches (e.g., AMALI's whole-kernel averaging).

7.8 Threats to Validity

Our venue-focused search may under-represent industry publications. We exclude proprietary tools (Nsight Compute [56], internal TPU models) from evaluation. Without GPU hardware, D1 relies on self-reported accuracy—independent verification would strengthen these findings. Our evaluation covers 5 of 22 tools, selected for methodology diversity; a complete study would include SimAI, AMALI, and Habitat. MTAP weights reflect our assessment of practitioner priorities; alternative weightings would shift relative rankings.

8 Open Challenges and Future Directions

Our MTAP evaluation (Sections 6–7) exposes five concrete research directions, each grounded in empirical gaps.

1. Bridging the composition gap. The composition problem (Figure 6) is the field's most pressing unsolved challenge. Kernel-level errors of 2–3% yield ~5–12% model-level error through uncaptured inter-kernel overheads ($\sigma_{\text{model}} \approx \sigma_{\text{kernel}} \cdot \sqrt{N}$ for uncorrelated errors, compounding linearly when correlated). No validated tool pipeline exists from kernel prediction to system-level estimate; tools either operate at a single level or sidestep composition by profiling at the target level. Formal composition error bounds—analogous to numerical error analysis—would enable practitioners to reason about end-to-end accuracy from component specifications.

2. Frontier workload coverage. The temporal validation lag (Section 4) is closing for transformers but remains wide: MoE, diffusion [40], and dynamic inference lack validated tools; scaling laws [14, 22, 27, 37] predict loss but not latency. Figure 7 shows the post-2023 shift toward LLM workloads.

3. Hardware transfer and emerging architectures. Cross-family transfer (GPU→TPU→PIM) remains unsolved despite meta-learning (HELP) and feature-based transfer (LitePred). PIM [26, 31, 46, 58], chiplets, and disaggregated designs blur memory hierarchy assumptions that current analytical models rely on.

4. Standardized evaluation infrastructure. No MLPerf [53, 68] equivalent exists for performance *prediction*. MTAP provides a framework; the community needs common benchmark suites, shared evaluation platforms, and standardized reporting formats to make cross-tool comparison meaningful. Portable workload formats (ONNX, Chakra [74]) and Docker-first deployment are prerequisites.

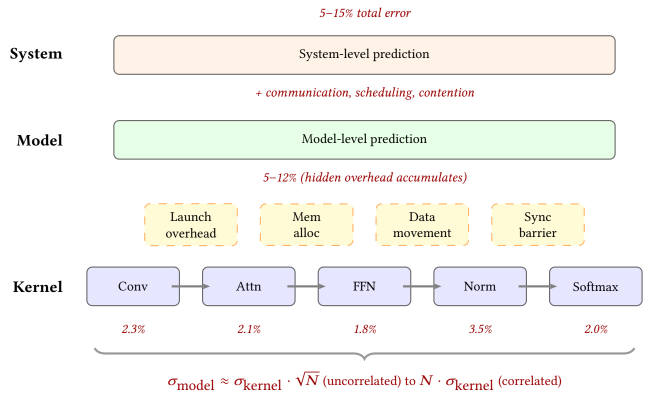
5–15% total error
System
System-level prediction
+ communication, scheduling, contention
Model
Model-level prediction
5–12% (hidden overhead accumulates)
Launch overhead
Mem alloc
Data movement
Sync barrier
Kernel
Conv
Attn
FFN
Norm
Softmax
2.3%
2.1%
1.8%
3.5%
2.0%
$\sigma_{\text{model}} \approx \sigma_{\text{kernel}} \cdot \sqrt{N}$ (uncorrelated) to $N \cdot \sigma_{\text{kernel}}$ (correlated)

Figure 6: Error composition across abstraction levels. Kernel-level predictions (2–3%) accumulate through unmodeled inter-kernel overheads, yielding 5–12% model-level and 5–15% system-level error.

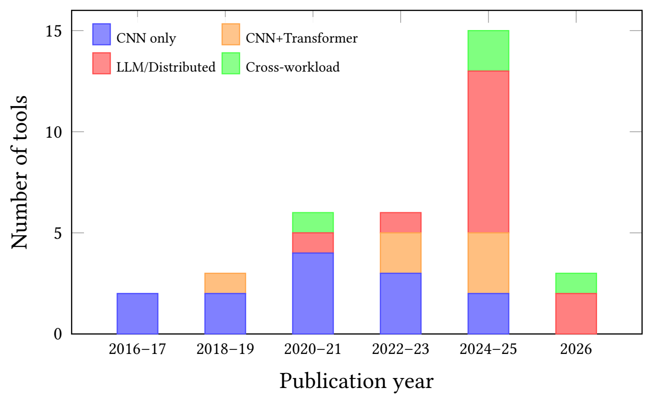

Figure 7: Workload coverage by publication period. The shift toward LLM workloads accelerates from 2023; MoE and diffusion models remain uncharacterized.

5. Temporal stability. Software stack evolution (FlashAttention [16], new CUDA versions, framework updates) silently invalidates models. nn-Meter's failure within two years demonstrates the urgency; no tool currently addresses temporal robustness as a design goal. Future tools should adopt continuous validation against evolving baselines [67].

9 Conclusion

This survey of 22 ML performance modeling tools introduces the Multi-dimensional Tool Assessment Protocol (MTAP), a principled evaluation framework that goes beyond accuracy to assess compositional fidelity, generalization robustness, deployment viability, and extensibility. Applying MTAP to five tools yields three actionable findings. First, *structural decomposition aligned with hardware execution boundaries* is the dominant design principle: Timeloop's loop nests for systolic arrays, NeuSight's tiles for GPU SMs, and VIDUR's prefill/decode phases all succeed by matching prediction granularity to hardware scheduling units. Second, *the composition gap is the field's central unsolved problem*: kernel-level errors (2–3%) amplify by 5–10× at the system level through unmodeled inter-kernel overheads, and no tool provides formal composition guarantees. Third, *deployment methodology predicts usability better than modeling sophistication*: Docker-first tools remain usable years later, while the tool with the lowest reported error (nn-Meter, <1%) fails to produce any output. The most pressing needs are standardized evaluation infrastructure (adopting MTAP or similar frameworks), validated tools for frontier workloads, and formal error composition bounds.

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] 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

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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).

[75] 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..

[76] 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

[77] 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.

[78] 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.

[79] 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.

[80] 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.

[81] 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..

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

[83] 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

[84] 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

[85] 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

[86] 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.

[87] 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.

[88] 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

[89] 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.

[90] 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.

[91] 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.

[92] 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.