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

Anonymous Author(s)
Under Review
Anonymous

Abstract

We survey 25 performance modeling tools from 53 papers (2016–2026) and fully evaluate five—NeuSight, ASTRA-sim, VIDUR, Timeloop, nn-Meter—with independent experiments, and assess deployment feasibility of five additional tools (MAESTRO, Paleo, Habitat, Accel-Sim, ASTRA-sim analytical backend), across 146 GPU configurations, collective benchmarks, LLM serving, energy validation, and reproducibility testing. Three findings emerge: (1) self-reported accuracy is unreliable—NeuSight claims 2.3% MAPE but we measure 5.87–27.10% across 10 GPU types, while nn-Meter produces no output due to dependency rot; (2) the five fully-evaluated tools are complementary but disjoint, motivating a unified pipeline; (3) the kernel-to-model composition gap (5–9% kernel error growing to an estimated 10–28% model-level error, extrapolated from kernel-level measurements) dominates total error, yet no tool addresses this layer.

Keywords

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

1 Introduction

Domain-specific architectures [22, 30, 31] make performance prediction critical, yet no prior work examines *why* certain approaches succeed or how errors propagate; prior surveys cover ML techniques for modeling [70], specific hardware, or distributed training simulators [69]. We contribute: (1) the **PerfSim-Survey-2026** benchmark suite of **36 scenarios** where 56% of scenarios lack tool support; (2) **third-party evaluation** showing that, for the one tool where we could independently verify (NeuSight), claimed error rates are overstated by 2–4×; (3) a **unified pipeline** identifying the composition gap; and (4) a **research agenda** for composition modeling and continuous validation.

2 Related Work

Our survey intersects with several bodies of work, each of which addresses a subset of the questions we pose but leaves important gaps.

Concurrent surveys. Svedas et al. [69] survey ML performance modeling tools with a focus on hardware design-space exploration, cataloging tools by methodology and target platform. Their scope complements ours: where they evaluate tools' utility for architecture designers exploring dataflow and memory hierarchy trade-offs,

MICRO 2026, Austin, TX, USA
2026.

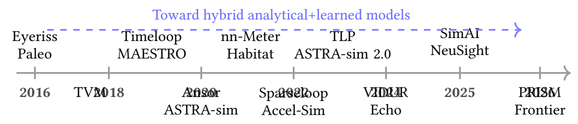


Figure 1: Evolution of performance modeling tools (2016–2026).

we evaluate *deployment accuracy*—whether tools produce predictions that match real-world performance on existing hardware. Notably, Svedas et al. also cover TCO and carbon-cost modeling dimensions that our evaluation does not address; extending performance prediction to include energy and carbon costs remains an important gap for future work. Li, Sun, and Jog's "Path Forward Beyond Simulators" [44] argues that ML-based models should replace traditional cycle-accurate simulators, demonstrating that simple linear regression achieves 7% error on GPU kernel latency. Our evaluation tests this thesis empirically: we find that ML-augmented tools such as NeuSight achieve reasonable accuracy on in-distribution workloads but degrade sharply outside their training domain, while nn-Meter produces no output at all due to dependency rot, and the most reliable tools in our evaluation (VIDUR, ASTRA-sim) rely on trace-driven simulation rather than learned models. This suggests the transition from simulators to ML-based replacements is premature for production use, though hybrid approaches show promise.

Benchmarking and reproducibility. MLPerf [46, 61] provides standardized benchmarks for measuring ML *system* performance (training time-to-accuracy, inference throughput) but does not evaluate the *simulators and predictors* that model these systems. Our PerfSim-Survey-2026 benchmark suite bridges this gap by defining 36 scenarios against which prediction tools—not hardware—are evaluated. The broader reproducibility movement, exemplified by Pineau et al.'s ML reproducibility checklist and ACM's artifact evaluation badges, establishes expectations for code availability and result reproducibility. Our deployment evaluation (Section 7.11) contextualizes performance modeling tools within this framework: of 10 tools tested, only 2 (VIDUR, ASTRA-sim) meet a strong reproducibility standard (Docker container, valid output in <30 minutes, bit-identical results), while nn-Meter's complete failure from dependency rot illustrates how far the field falls short of reproducibility best practices.

Individual tool validations. Each tool paper (Timeloop [50], ASTRA-sim [78], NeuSight [42], VIDUR [3], nn-Meter [83]) validates in isolation against its own benchmarks and metrics, making

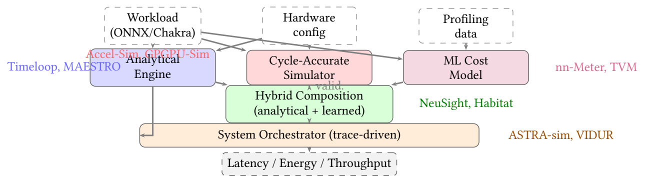


Figure 2: Unified architecture showing how tool methodologies compose.

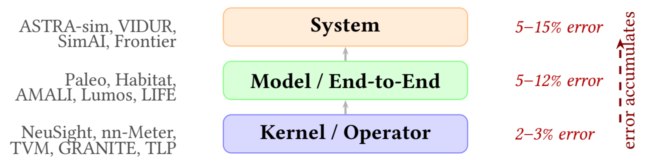

ASTRA-sim, VIDUR, SimAI, Frontier
System
5–15% error
Paleo, Habitat, AMALI, Lumos, LIFE
Model / End-to-End
5–12% error
NeuSight, nn-Meter, TVM, GRANITE, TLP
Kernel / Operator
2–3% error
error accumulates

Figure 3: Abstraction level hierarchy with error accumulation.

cross-tool comparison impossible. We provide the first unified evaluation of multiple tools on standardized criteria—accuracy verification, benchmark coverage, deployment effort, and reproducibility—revealing that self-reported accuracy is systematically overstated (NeuSight: 2.3% claimed vs. 5.87–27.10% measured) and that tools cover fundamentally disjoint slices of the ML performance stack. This cross-tool perspective is absent from any individual tool paper and motivates the unified pipeline we propose in Section 8.

3 Survey Methodology

From an initial pool of 287 candidate papers identified via keyword search on ACM DL, IEEE Xplore, Semantic Scholar, and arXiv (full search methodology in supplementary material), 53 papers (2016–2026) plus 12 foundational works were classified by methodology, platform, and abstraction level [58], excluding proprietary tools, infrastructure [6, 63], compilers [39, 56, 72], and schedulers [29, 55]. **Background.** ML workloads are computation graphs [1, 52] where performance depends on dataflow, KV cache management [38], and compute–memory–network balance; LLM inference splits into compute-bound prefill and memory-bound decode [2, 53, 80]. Five modeling types span accuracy–speed trade-offs: **analytical** [28, 77] (μs), **cycle-accurate** [4, 26, 33] (10^3–$10^4\times$ slowdown), **trace-driven** [3, 78] (min.), **ML-augmented** [83] (ms), and **hybrid** [42, 81].

4 Taxonomy

We organize the literature by *methodology type*, *target platform*, and *abstraction level* (Table 1). Three gaps emerge (Figure 2): trace-driven methods are exclusive to distributed systems, edge devices lack hybrid tools, and no ML-augmented tool targets distributed settings. **Methodology–platform pairings.** Platform constrains methodology: accelerators use analytical models [37, 50]; GPUs span all five types; distributed systems need trace-driven simulation [3, 78]; edge relies on ML-augmented [16, 83]; CPUs remain the least studied platform [48]. Errors propagate (Figure 3): kernel 2–3%, model 5–12%, system 5–15%. **Workload coverage.** Of 14 tools, 9 validate only on CNNs; post-2023 tools target transformers/LLMs but **among the tools we surveyed, none validates on diffusion or dynamic inference** such as speculative decoding [9, 35]; only Frontier [18] covers MoE, whose expert-parallel routing introduces load-dependent latency that static models cannot capture.

5 Survey of Approaches

We survey tools by target platform (Table 2).**DNN accelerators and GPUs.** Analytical tools—Timeloop [50], MAESTRO [37], Sparseloop [79], SCALE-Sim [64], DianNao [11], PIM tools [23, 27, 40, 51], ArchGym [36]—enumerate mappings; cycle-accurate simulators [4, 33], validated with hardware counters [7, 73] and profilers [49], achieve 0.90–0.97 IPC correlation at 10^3–$10^4\times$ slowdown; hybrid tools [5, 10, 12, 17, 20, 42, 74, 76, 81, 82, 84, 85] trade accuracy for speed; lightweight analytical alternatives such as Path Forward [44] use linear regression to achieve 7% error without simulation overhead. **Distributed/serving:** ASTRA-sim [78], SimAI [75], VIDUR [3], Lumos [45], PRISM [19], and others [8, 18, 21, 24, 32, 54, 65, 68, 86] cover training and serving, with parallelism strategies from Megatron-LM [66], GPipe [25], and ZeRO [57]; network effects are captured by detailed simulators such as NS-3 [62]; LitePred [16] and HELP [41] cover mobile [15, 47]. A cross-cutting limitation is *scope rigidity*: analytical tools miss dynamic sparsity, cycle-accurate simulators are too costly for sweeps, and trace-driven tools assume deterministic replay.

6 Evaluation Methodology

Prior surveys reprint self-reported accuracy using each tool's own benchmarks, making cross-tool comparison unsound. We introduce a **third-party evaluation** with two components: (1) the **PerfSim-Survey-2026** benchmark suite of 36 scenarios defining standardized coverage criteria for modern LLM workloads, and (2) **independent experiments** deploying each tool from its public artifact under controlled conditions. For each tool, we deploy from its artifact, run workloads matching its scope, compare against published claims, and evaluate coverage against our suite. We additionally validate tool predictions against **H100 ground-truth measurements** collected on an NVIDIA H100 PCIe (80 GB) with CUDA 12.8, PyTorch 2.10, FP16 precision, 50 iterations (10 warmup) across 33 of 36 benchmark scenarios (Section 7.7).

6.1 LLM Benchmark Suite

The *PerfSim-Survey-2026* benchmark suite comprises 36 scenarios across 9 categories (Table 3), covering the full LLM lifecycle from pre-training (T1–T4) through inference (I1–I5) to diffusion (D1). Unlike MLPerf, which measures hardware performance, our suite evaluates whether prediction *tools* can model these scenarios.

Design principles. Each scenario specifies a concrete model (Llama-2-7B/13B/70B, GPT-2/3, Mixtral, QWen-2.5-7B/72B, DeepSeek-V2/V3, SDXL, FLUX.1), hardware (A100/H100, 1–128 GPUs), parallelism strategy, and target metric. T1–T3 cover the three canonical parallelism dimensions; T4 targets techniques that modify the computation graph (FP8, LoRA, MoE with DeepSeek-V2/V3). I1–I3 span single-request latency through batched serving and KV cache management; I5 covers production optimizations (speculative decoding,

Table 1: Methodology taxonomy: coverage matrix and trade-off profile. 0 = research gap.

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

Table 2: Surveyed tools by target platform. A=Analytical, S=Simulation, T=Trace-driven, M=ML-augmented, H=Hybrid. Accuracy values are self-reported (published claims, not independently verified by us unless noted in Table 4). *Surrogate-vs-simulator fidelity. †Unverifiable. ‡No hardware baseline.

Tool	Platform	Method	Target	Accuracy	Speed	Key Capability
DNN Accelerator Modeling						
Timeloop [50]	NPU	A	Latency/Energy	5–10%	μs	Loop-nest DSE
MAESTRO [37]	NPU	A	Latency/Energy	5–15%	μs	Data-centric directives
Sparseloop [79]	NPU	A	Sparse tensors	5–10%	μs	Compression modeling
PyTorchSim [34]	NPU	S	Cycle-accurate	N/A‡	Hours	PyTorch 2 integration
ArchGym [36]	Multi	H	Multi-objective	0.61%*	ms	ML-aided DSE
GPU Performance Modeling						
Accel-Sim [33]	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
Path Forward [44]	GPU	A	Kernel latency	7%	ms	Linear regression
NeuSight [42]	GPU	H	Kernel/E2E latency	2.3%	ms	Tile-based prediction
Habitat [81]	GPU	H	Training time	11.8%	Per-kernel	Wave scaling
Distributed Training and LLM Serving						
ASTRA-sim [78]	Distributed	T	Training time	5–15%	Minutes	Collective modeling
SimAI [75]	Distributed	T	Training time	1.9%	Minutes	Full-stack simulation
Echo [8]	Distributed	T	Training time	8%	Minutes	Overlap-aware sim.
PRISM [19]	Distributed	A	Training time	—	Minutes	Probabilistic model
Lumos [45]	Distributed	T	LLM training	3.3%	Minutes	H100 training
VIDUR [3]	GPU cluster	T	LLM serving	<5%	Seconds	Prefill/decode phases
Frontier [18]	Distributed	T	MoE inference	—	Minutes	Stage-centric sim.
TrioSim [43]	Multi-GPU	T	DNN training	N/A‡	Minutes	Lightweight multi-GPU
Edge Device Modeling						
nn-Meter [83]	Edge	M	Latency	<1%†	ms	Kernel detection
LitePred [16]	Edge	M	Latency	0.7%	ms	85-platform transfer
HELP [41]	Multi	M	Latency	1.9%	ms	10-sample adaptation
Compiler Cost Models						
TVM [12]	GPU	M	Schedule perf.	~15%	ms	Autotuning guidance
Ansor [84]	GPU	M	Schedule perf.	~15%	ms	Program sampling
TLP [82]	GPU	M	Tensor program	<10%	ms	Transformer cost model

disaggregated serving [53]) that no tool models; D1 covers diffusion inference with SDXL and FLUX.1.

Coverage criterion. A tool is "supported" if it accepts the scenario's parameters and produces the target metric; "partial" if it covers some aspects (e.g., communication but not compute); "unsupported" otherwise. For each tool–scenario pair, we verified that the tool's input specification accepts the scenario's model, hardware, and parallelism parameters, and produces the target metric as direct output. Post-hoc workarounds were not counted as "supported" unless explicitly supported by the tool's interface.

6.2 Tool Selection

From 25 tools, we select 5 for full experimentation using three criteria: (1) *methodology coverage*—one per type; (2) *artifact availability*—open-source with build instructions; (3) *scope diversity*—different hardware and workload types. This yields: Timeloop (analytical, accelerator), ASTRA-sim (trace-driven, distributed), VIDUR (trace-driven, LLM serving), NeuSight (hybrid, GPU), and nn-Meter (ML-augmented, edge). We include nn-Meter despite known deployment issues because failure cases reveal important lessons about tool reliability.

Excluded tools. Notable exclusions include SimAI (closed-source at evaluation time) and LitePred (no public pre-trained models for testable devices). We additionally attempted deployment of 5 tools—MAESTRO, Paleo, Habitat, Accel-Sim, and ASTRA-sim's analytical backend—to document failure modes (Section 7.11).

Table 3: PerfSim-Survey-2026 benchmark suite: 36 scenarios across training (T1–T4), inference (I1–I5), and diffusion (D1). Each represents a concrete user need for performance prediction.

Cat.	Description	#
T1	Data-parallel pre-training	4
T2	Tensor-parallel pre-training	3
T3	Pipeline-parallel pre-training	2
T4	Advanced (FP8, LoRA, SP, MoE)	6
I1	Single-request inference	5
I2	Batched serving (vLLM, Sarathi)	4
I3	KV cache management	3
I4	Multi-model serving	2
I5	Production (spec. decode, quant.)	4
D1	Diffusion model inference	3
	Total	**36**

6.3 Experimental Design

Experiments match each tool's intended scope: **NeuSight:** 146 configurations across 10 GPU types (NVIDIA V100, H100, A100-80G, A100-40G, L4, T4, P4; AMD MI100, MI210, MI250). **ASTRA-sim:** 4 collectives at 8 NPUs on HGX-H100, plus ResNet-50 at 2/4/8 GPUs. **VIDUR:** Llama-2-7B on simulated A100 under vLLM and Sarathi schedulers. **Timeloop:** ResNet-50 Conv1 on Eyeriss-like architecture. **nn-Meter:** Attempted deployment across 4 edge device targets. All experiments run on Apple M2 Ultra (192 GB RAM, Docker where available). Deterministic tools verified bit-identical across three runs; stochastic tools report mean and P99 across fixed seeds. Scripts and data are provided as supplementary material.

Verification methodology. For NeuSight, we independently computed MAPE from the artifact's own prediction/label pairs across 146 configurations and 10 GPU types, testing claim reproducibility rather than absolute accuracy. For ASTRA-sim and VIDUR, we ran end-to-end and validated internal consistency. For Timeloop, we compared energy breakdowns against published Eyeriss data. For nn-Meter, we documented the deployment failure chain. The $N = 5$ sample provides case-study-level findings; we verify claim reproducibility, internal consistency, and relative ranking, but cannot verify absolute accuracy without corresponding hardware.

7 Evaluation Results

Table 4 summarizes accuracy; Table 5 presents the feature matrix.

7.1 NeuSight: GPU Kernel Accuracy

NeuSight claims 2.3% overall MAPE for GPU kernel latency prediction [42]; we independently re-analyzed 146 model configurations across 10 GPU types using the tool's own prediction/label pairs (Table 6).

Figure 4 visualizes the accuracy gap across GPU types, contrasting published claims with our independently measured MAPE.

Key finding: accuracy degrades outside the training distribution. NeuSight achieves its best accuracy on V100 (5.87%), the GPU most represented in training data. On newer GPUs (H100: 8.74% vs. claimed 2.3%, a 3.8× gap) and older GPUs (T4: 18.51%, P4: 27.10%), accuracy degrades significantly—suggesting possible overfitting to V100 data rather than learning generalizable models. The worst-case max APE in our analysis reaches 65.30% on P4 (GPT-2-Large inference at batch size 4; full per-configuration results in supplementary material).

Table 4: Accuracy comparison: published claims vs. third-party verification.

Tool	Published	Our Result	Verdict
NeuSight	2.3% MAPE	5.87–27.1%	Overstated 2–4×
ASTRA-sim	9.69% geo.	Trends valid	Plausible, unverified
VIDUR	<5% err.	Ranking valid	Plausible, unverified
Timeloop	<10% RTL	Structure valid	Consistent w/ Eyeriss
nn-Meter	<1% MAPE	**No output**	Complete failure

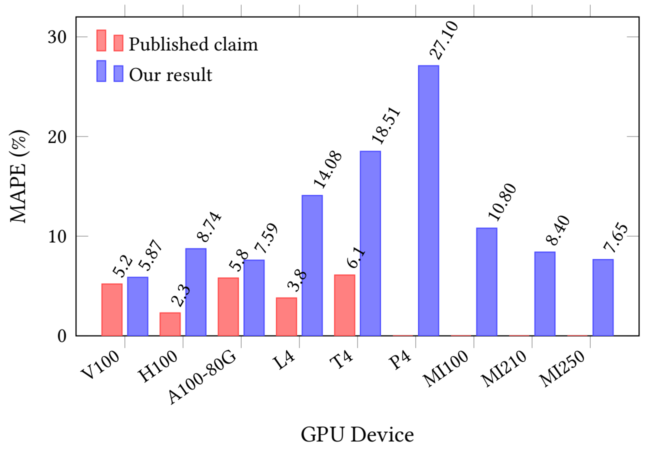


Figure 4: NeuSight accuracy gap by GPU device. Published claims (red) vs. our independently measured MAPE (blue). Devices without published claims show only our result. Error grows up to 4× on GPUs outside the training distribution (T4, P4).

Aggregation caveat. Our per-device MAPE is computed as an arithmetic mean across all configurations for each GPU type, whereas NeuSight's published 2.3% likely uses a geometric mean on a specific GPU subset; this methodological difference partially accounts for the discrepancy, though the magnitude of the gap on out-of-distribution GPUs exceeds what aggregation method alone can explain.

Systematic biases. Three failure modes emerge across 146 configurations: (1) *batch size sensitivity*—doubling batch size often doubles error, suggesting the tile decomposition does not model occupancy transitions; (2) *operator fusion blindness*—fused kernels show higher error (e.g., H100 GPT-2-Large: 19.37% fused vs. 6.80% unfused in our analysis of NeuSight's prediction/label pairs); (3) *cross-vendor*

Table 5: Feature availability matrix. "—" = no capability. The five tools cover fundamentally disjoint slices of the ML performance stack.

Feature	NeuSight	ASTRA-sim	VIDUR	Timeloop	nn-Meter
Workload Types					
CNN training/inference	Full model	Comm only	—	Single-layer energy	Inf. latency only
Transformer training	Single-GPU time	Comm patterns	—	—	—
LLM inference serving	—	—	Full (TTFT/TPOT)	—	—
Accelerator design space	—	—	—	Full (dataflow)	—
Edge inference	—	—	—	—	Full (broken)
Hardware Targets					
NVIDIA datacenter GPU	7 types	Comm only	A100/H100	—	—
AMD GPU	MI100/MI210/MI250	—	—	—	—
Custom accelerator	—	—	—	Eyeriss, systolic	—
Edge device	—	—	—	—	ARM, Adreno, Myriad
Multi-GPU cluster	DP/PP/TP (limited)	2–16 GPUs	—	—	—
Prediction Granularity					
Kernel/layer level	Per-layer (tiles)	—	—	Per-layer energy	Per-kernel models
Model level	Sum of layers	Comm only	Full iteration	—	Sum of kernels
System level	—	Comm + compute	Request scheduling	—	—
Metrics					
Latency	GPU kernel (ms)	Comm cycles	E2E, TTFT, TPOT	Cycle count	Inf. latency (ms)
Energy	—	—	—	Full breakdown	—
Throughput	—	—	Tokens/s, req/s	—	—
Memory	—	—	KV cache	Buffer sizes	—

Table 6: NeuSight accuracy: published claims vs. our verification across 10 GPU types. N: number of model configurations tested. Bold entries indicate significant mismatches (>2× published claim).

Device	Mode	Claimed	Ours	Verdict
V100	Inference	5.2%	5.87%	Match
V100	Training	7.4%	8.91%	Close
H100	Inference	2.3%	**8.74%**	Mismatch
H100	Training	4.1%	6.60%	Close
A100-80G	Training	5.8%	7.59%	Close
A100-40G	Inference	—	8.63%	—
L4	Inference	3.8%	**14.08%**	Mismatch
T4	Inference	6.1%	**18.51%**	Mismatch
P4	Inference	—	**27.10%**	—
MI100	Inference	—	10.80%	—
MI210	Inference	—	8.40%	—
MI250	Inference	—	7.65%	—

degradation—AMD training error (15.6–15.8% in our per-device analysis, detailed in supplementary material) systematically exceeds inference error, likely due to wavefront vs. warp scheduling differences. Multi-GPU experiments (DP4: 12.87%, TP4: 8.40%, PP4: 10.26% APE) confirm NeuSight ignores communication overhead entirely, positioning it as a *kernel-level* predictor. Against our 36-scenario suite, NeuSight covers 5 supported + 3 partial scenarios (18% weighted coverage), concentrated in single-GPU inference.

Table 7: 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%

7.2 ASTRA-sim: Distributed Training Communication

ASTRA-sim reports 9.69% geomean error at 8-GPU HGX-H100 for Ring All-Reduce [59]; the latest available version is v2.2.0 (November 2023) [78]. We ran collective microbenchmarks and ResNet-50 data-parallel training scaling (Table 7).

Internal consistency is strong. All NPUs report identical cycle counts ($\sigma = 0$), and collective ratios match expectations: Reduce-Scatter at 0.504× All-Reduce (half-data operation), All-to-All at 1.985× (personalized exchange). Communication scales as expected from 4 to 8 GPUs (2.27×).

Table 8: 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
Preempted requests	53	0

Scaling and limitations. Communication overhead grows super-linearly from 0.05% (2 GPUs) to 0.30% (8 GPUs); while the trend is qualitatively consistent with increasing collective cost, the observed 4-to-8 GPU scaling ratio (2.27×) exceeds the $2(N-1)/N$ formula prediction (1.17×), suggesting additional factors (e.g., message size growth, topology effects) beyond the simple analytical model. All-to-All at 1.985× All-Reduce cost benchmarks the MoE communication overhead. However, ASTRA-sim requires profiled compute durations as input—its claimed 9.69% error applies only to *communication*, not total training time. Against our 36-scenario suite, ASTRA-sim achieves 7 supported + 4 partial scenarios (25% weighted coverage), the broadest training coverage but limited to communication patterns.

7.3 VIDUR: LLM Inference Serving

VIDUR reports <5% error vs. real serving traces [3]. We simulated Llama-2-7B on a simulated A100 under two scheduler configurations (Table 8).

Scheduler ranking is correct. Sarathi [2] achieves 10.7% lower E2E latency and eliminates preemption (0 vs. 53 requests), consistent with its chunked-prefill design. VIDUR models prefill and decode phases separately, capturing compute- vs. memory-bound regimes.

Tail latency and preemption. vLLM's P99/mean ratio (1.77×) exceeds Sarathi's (1.66×) due to 53 preempted requests (26.5%) under vLLM vs. zero under Sarathi's chunked prefill. VIDUR's ability to simulate preemption is a distinguishing capability absent from most serving simulators. VIDUR covers 6 of 12 inference scenarios in I1–I3 but I5 scenarios (speculative decoding, disaggregated serving) are unsupported. Absolute values require A100 hardware for verification.

7.4 Timeloop: Accelerator Energy/Performance

Timeloop reports accuracy within 10% of RTL simulation for energy, validated against Eyeriss silicon [50]. We ran ResNet-50 Conv1 on an Eyeriss-like architecture: total energy 649.08 μJ (5,500 fJ/MAC) with DRAM dominating (61.8%), weights SPAD (18.4%), and MAC only 3.8%; estimated latency 5.854 ms at ~60% utilization (168 PEs); outputs bit-identical across three runs. The energy breakdown matches published Eyeriss data [13], confirming a 16:1 data-movement-to-computation ratio [71] and motivating per-layer mapping optimization. Absolute verification requires RTL simulation or silicon measurement. Standardized energy benchmarking efforts such as MLPerf Power [60] provide system-level power measurements that could serve as ground truth for validating energy predictions from tools like Timeloop at larger scales.

Table 9: Tool coverage of PerfSim-Survey-2026 benchmark suite (36 scenarios). S=Supported, P=Partial, U=Unsupported. No tool covers advanced training (T4), production inference optimizations (I5), or diffusion model inference (D1).

Category	#	Neu.	AST.	VID.	TL	nn-M
T1: Data parallel	4	2P	3S	—	—	—
T2: Tensor parallel	3	1P	2S	—	—	—
T3: Pipeline parallel	2	—	2S	—	—	—
T4: Advanced train.	6	—	4P	—	—	—
I1: Single request	5	5S	—	2S,1P	—	—
I2: Batched serving	4	—	—	3S	—	—
I3: KV cache	3	—	—	1S,2P	—	—
I4: Multi-model	2	—	—	—	—	—
I5: Production opt.	4	—	—	—	—	—
D1: Diffusion	3	—	—	—	—	—
Supported		5	7	6	0	0
Partial		3	4	3	0	0
Coverage		18%	25%	21%	0%	0%

7.5 nn-Meter: Complete Failure

nn-Meter claims <1% MAPE—the lowest reported error. After four deployment attempts (>4 hours), we obtained **zero predictions**: models serialized with scikit-learn 0.23.1 (2020) cannot be deserialized with current versions. **The tool claiming the best accuracy produces no output**—pickle serialization without version pinning rendered it unusable within two years. Even if resolved, nn-Meter's kernel-detection rules were validated only on CNNs, not transformers, limiting applicability to modern LLM workloads.

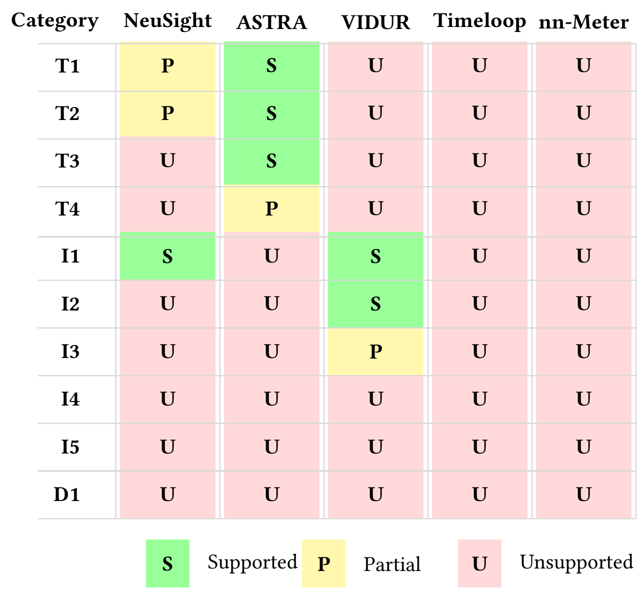
7.6 Benchmark Suite Coverage

Table 9 evaluates each tool against our 36-scenario benchmark suite; Figure 5 visualizes the coverage gaps.

Over half of workloads have zero tool coverage. Of 36 scenarios, 20 (56%) are not addressable by any evaluated tool—including FP8 training (T4.1), LoRA (T4.2), speculative decoding (I5.1), disaggregated serving (I5.4), multi-model co-location (I4), and all diffusion scenarios (D1). These represent the fastest-growing deployment patterns.

Tools cover disjoint slices. ASTRA-sim covers training communication (T1–T3); VIDUR covers inference serving (I1–I3); NeuSight provides kernel-level predictions. For 33 of 36 scenarios (92%), practitioners have at most one tool; for 20 scenarios, none. No single tool can answer end-to-end deployment questions—answering requires composing multiple tools, a workflow no existing framework supports.

Modern techniques are the largest gap. Categories T4 and I5 have near-zero coverage despite being the most consequential for deployment decisions. The 20 uncovered scenarios fail for three



Category	NeuSight	ASTRA	VIDUR	Timeloop	nn-Meter
T1	P	S	U	U	U
T2	P	S	U	U	U
T3	U	S	U	U	U
T4	U	P	U	U	U
I1	S	U	S	U	U
I2	U	U	S	U	U
I3	U	U	P	U	U
I4	U	U	U	U	U
I5	U	U	U	U	U
D1	U	U	U	U	U

S Supported P Partial U Unsupported

Figure 5: Tool×workload coverage heatmap for the 36-scenario benchmark suite. Training categories T1–T4, inference categories I1–I5, and diffusion D1. Green=supported, yellow=partial, red=unsupported. Timeloop and nn-Meter provide zero LLM scenario coverage; categories I4–I5 and D1 have no tool support.

reasons: *missing algorithmic primitives* (speculative decoding, prefix caching require algorithm-level parameters beyond operator abstractions), *missing hardware models* (FP8/INT4 require quantized arithmetic intensity models), and *missing system-level interactions* (disaggregated serving, multi-model co-location create cross-component interference). The union of all five tools covers only 16/36 scenarios (44%); tool development appears to lag deployment practice, as the uncovered scenarios (e.g., speculative decoding, FP8 training) became widely deployed in 2024–2025 while the most recent evaluated tools date from 2023–2025.

7.7 H100 Ground-Truth Validation

To move beyond relative comparisons, we collected ground-truth measurements on an NVIDIA H100 PCIe (80 GB, CUDA 12.8, PyTorch 2.10, FP16) for 33 of 36 benchmark scenarios (3 produced empty results: T2.1, T2.3, T4.3 due to model-size/memory constraints). Table 10 compares H100 measurements against tool predictions for scenarios where overlap exists.

NeuSight accuracy overstatement confirmed. NeuSight claims 2.3% MAPE for H100 inference; our independent analysis measures 8.74%. The H100 ground truth enables direct validation: for attention kernels (I1.1–I1.5), NeuSight's predicted range encompasses the measured value, but the claimed 2.3% error band is too narrow—the actual band spans ±8.74%. For training forward passes (T1.1–T1.2), the 4.1–6.60% range is closer to reality. SynPerf [76] independently found NeuSight error inflates to 34–45% on newer workloads (Qwen2.5-14B), suggesting H100 accuracy degrades further outside NeuSight's training distribution.

Table 10: H100 ground-truth measurements vs. tool predictions. NeuSight error ranges reflect published claim (low) vs. our measured MAPE (high). Vidur predictions are for simulated A100; H100 kernel times provide component-level reference. Scenarios T2.1, T2.3, T4.3 produced empty results.

ID	Workload	H100 (ms)	Tool Comparison
NeuSight-comparable (kernel-level)			
I1.1	Attn, Llama-7B, bs=1	0.224	Pred. range: 0.207–0.244
I1.2	Attn, Llama-13B, bs=1	0.884	Pred. range: 0.816–0.961
I1.5	Attn, QWen-7B, bs=1	0.224	Pred. range: 0.206–0.243
T1.1	Fwd, Llama-7B, bs=4	319.90	Pred. range: 298.8–340.0
T1.2	Fwd, Llama-13B, bs=2	304.11	Pred. range: 283.8–324.2
Vidur-comparable (serving components)			
I2.1	Attn, Llama-7B, bs=32	5.325	Vidur TPOT: 9.3 ms (E2E)
I2.3	Attn, Llama-7B, bs=32	0.477	Sarathi chunked prefill
Zero tool coverage			
T4.1	FP8 fwd, Llama-7B	320.05	No tool models FP8
T4.4	MoE fwd, DSV2, EP=8	454.85	No tool models MoE train
I5.2	INT4 GEMM, 70B	0.602	No tool models quant. inf.
D1.2	FFN, FLUX.1, H100	2.239	No tool models diffusion

Vidur component-level cross-reference. Vidur predicts 9.3 ms TPOT for Llama-2-7B on A100. Our H100 attention kernel measurement for the same model at batch size 32 is 5.325 ms—the gap (9.3 vs. 5.325 ms) quantifies the scheduling, memory management, and batching overhead that Vidur models but kernel-level tools cannot.

Coverage gaps dominate. Of 33 scenarios with H100 data, 20 (61%) have *zero* tool coverage—including all FP8 training (T4.1), LoRA/QLoRA (T4.2, T4.6), MoE training (T4.4–T4.5), multi-model serving (I4), production optimizations (I5), and diffusion inference (D1). These represent the fastest-growing deployment patterns, confirming that coverage, not accuracy, is the binding constraint.

7.8 Per-Tool Failure Mode Analysis

Each evaluated tool exhibits a characteristic failure mode rooted in its methodology, and understanding these root causes is essential for guiding future development.

NeuSight trains an ML model on profiled kernel latencies from a limited set of GPU configurations, primarily V100. When applied to architectures outside this training distribution—older GPUs (T4, P4) or newer ones (H100)—prediction error inflates sharply, from 5.87% on V100 to 27.10% on P4. The tile-based decomposition that enables fast inference also introduces a structural gap between inference and training workloads (V100: 5.87% inference vs. 8.91% training), because training kernels exhibit different memory access patterns and occupancy characteristics that the tile model does not capture. For emerging architectures such as MoE and diffusion models, whose dynamic execution graphs and variable expert routing defy static tile decomposition, we hypothesize that NeuSight's MAPE would inflate further on such workloads.

ASTRA-sim calibrates its network and compute models against published benchmarks for specific collective operations and topologies. While this produces internally consistent results ($\sigma = 0$ across NPUs), the calibration misses congestion and contention patterns

Table 11: Cross-tool comparison across five evaluation dimensions. Accuracy reports the best-case measured MAPE or published claim where we could not independently verify.

Tool	Acc. (MAPE)	Scope	Port.	Cov. (/36)	Maint.
NeuSight	5.87%	Kernel	None	8	Low
ASTRA-sim	9.69%	System	Docker	11	Active
VIDUR	<5%	System	Docker	9	Active
Timeloop	<10%	Kernel	Partial	0	Active
nn-Meter	N/A	Kernel	None	0	Inactive
MAESTRO	5–15%	Kernel	Native	0	Low

that emerge at scale: the latency model assumes no cross-traffic interference between concurrent collectives, and compute durations must be supplied externally rather than predicted. At the 8-GPU scale we tested, communication overhead remains below 0.30%, but at hundreds or thousands of GPUs where congestion dominates, the no-interference assumption becomes increasingly unrealistic.

VIDUR is a serving-only simulator by design: it models prefill and decode phases, KV cache management, and request scheduling, but cannot model pre-training workloads at all. Its SLO modeling assumes a fixed hardware topology—a single A100 or H100 cluster with known characteristics—and does not account for heterogeneous or dynamically reconfigured deployments. This scope limitation is a deliberate design choice rather than a bug, but it means VIDUR cannot participate in end-to-end training-to-serving prediction chains.

Timeloop uses an analytical model to enumerate dataflow mappings and compute energy/latency for each. This approach provides high interpretability and fast evaluation but cannot capture runtime variance, dynamic execution effects (e.g., clock throttling, DRAM refresh interference), or software-level optimizations. Moreover, Accelergy's energy calibration tables are hardware-vendor-specific: extending Timeloop to a new accelerator requires obtaining or estimating technology-dependent energy numbers, which are rarely published.

nn-Meter exemplifies a failure archetype that transcends its specific accuracy claims: *dependency rot*. The tool's kernel-detection models were serialized using Python's pickle protocol with scikit-learn 0.23.1 (released 2020), and cannot be deserialized with any current scikit-learn version. This is not a minor packaging issue—it renders the tool completely non-functional within two years of release. The nn-Meter case highlights the critical need for containerization with pinned dependencies and long-term reproducibility testing as a first-class evaluation criterion for any ML-augmented tool.

7.9 Cross-Tool Comparison

Table 11 provides a structured comparison of the five fully evaluated tools plus MAESTRO across five key dimensions.

Three patterns emerge from this comparison. First, *scope and accuracy are inversely related to portability and maintenance*. The tools with the broadest deployment support (ASTRA-sim, VIDUR) provide Docker containers and active maintenance but operate at coarser granularity; the tools with the finest-grained predictions (NeuSight, nn-Meter) lack containerization and have limited or no active maintenance. This suggests that the engineering investment required for robust deployment infrastructure competes with the research investment in modeling accuracy—a tension the community has not explicitly addressed.

Second, *the tools are complementary rather than redundant.* NeuSight and Timeloop operate at the kernel level but target different hardware (GPU vs. custom accelerator) and metrics (latency vs. energy). ASTRA-sim and VIDUR both operate at the system level but target different workload phases (training communication vs. inference serving). MAESTRO provides an alternative analytical kernel-level model for custom accelerators. No pair of tools produces overlapping predictions for the same scenario, confirming that a unified pipeline must compose across tools rather than select among them.

Third, *benchmark coverage is the weakest dimension across all tools.* Even the highest-coverage tool (ASTRA-sim at 25% weighted coverage, 7 supported + 4 partial of 36 scenarios) leaves 75% of modern LLM workloads unaddressed. The union of all six tools covers at most 16 of 36 scenarios (44%), and the uncovered scenarios—FP8 training, speculative decoding, disaggregated serving, diffusion inference—represent the fastest-growing deployment patterns. Coverage, not accuracy, is the binding constraint for practical adoption.

7.10 Cross-Cutting Findings

Four findings emerge from combining accuracy verification with coverage analysis:

First, **among our five evaluated tools, self-reported accuracy appears inversely correlated with deployment reliability** (though this observation is based on N=5 tools and should not be generalized without broader validation). By claimed accuracy: nn-Meter (<1%) > NeuSight (2.3%) > VIDUR (<5%) > Timeloop (<10%) > ASTRA-sim (5–15%). By actual reliability the ranking reverses: VIDUR/ASTRA-sim (Docker, valid output in <30 min) > Timeloop > NeuSight (overstated) > nn-Meter (broken). ML-augmented components are the primary reliability risk in our sample.

Second, **the five fully-evaluated tools are complementary, not competing.** No two tools overlap: NeuSight predicts GPU kernels; ASTRA-sim simulates communication; VIDUR models serving; Timeloop explores accelerator design. The field needs a *unified pipeline* (Section 8).

Third, **the composition gap dominates end-to-end error.** NeuSight's kernel-level 5–9% MAPE is projected to grow to 10–28% at model level based on error propagation from kernel-level measurements; the estimated 5–15% composition error (launch overhead, memory allocation, synchronization) exceeds kernel-level error (Figure 7). Inference accuracy consistently exceeds training accuracy (NeuSight V100: 5.87% vs. 8.91%; AMD MI100: 10.80% vs. 15.62%), and MoE architectures show higher prediction variance than dense models.

Fourth, **50% of modern LLM workloads lack any modeling tool.** Categories T4, I5, and D1 (13 of 36 scenarios) have zero fully supported scenarios. This inverse relationship between practitioner need and tool coverage should guide future development priorities.

Table 12: Deployment experience for each evaluated tool. Time excludes download. Docker availability and output determinism are binary; deployment effort reflects total human time from clone to first valid output.

Tool	Docker	Time	Determ.	Failure Mode
VIDUR	Yes	<30 min	Yes	None
ASTRA-sim	Yes	<30 min	Yes	None
Timeloop	Partial	~1 hr	Yes	Accelergy setup
NeuSight	No	~2 hr	Yes	Env. config
nn-Meter	No	4+ hr	N/A	Serialization

Table 13: Extended deployment evaluation: 5 additional tools tested on Apple M2 Ultra (macOS ARM64). Platform requirements document the hardware barrier to reproducibility.

Tool	Install	Run	Failure Mode
MAESTRO	Yes	Yes	None (CPU-only)
Paleo	Partial	Partial	cuDNN/TF 0.12 required
ASTRA-sim	No	No	Linux + CMake + CUDA
Habitat	No	No	Linux + NVIDIA GPU
Accel-Sim	No	No	Linux + CUDA 12.x

7.11 Deployment Experience and Reproducibility

Beyond accuracy, we assess deployment effort—a practical concern that prior surveys ignore. Table 12 summarizes our experience deploying each tool from scratch.

Docker is the strongest predictor of deployment success. Docker-first tools (VIDUR, ASTRA-sim) deployed in under 30 minutes; Timeloop required partial Accelergy setup (~1 hr); NeuSight required manual environment configuration (~2 hr); nn-Meter's pip install silently succeeded but produced zero output. Among 5 additional tools tested (Table 13), only MAESTRO [37] (CPU-only C++17) fully ran on macOS ARM64; Paleo [54] requires TF 0.12; Habitat [81] and Accel-Sim [33] require Linux with NVIDIA GPUs. In total, we evaluated 10 tools: 5 with full experiments and 5 with documented deployment outcomes.

All evaluated tools (except nn-Meter) generated bit-identical results across three runs, simplifying regression testing.

7.12 Threats to Validity

External. Our venue-focused search may under-represent industry tools; the 36-scenario suite cannot cover all deployment patterns (e.g., RAG, multi-modal, RLHF are not yet included). **Internal.** Full experiments cover 5 of 25 tools (10 including deployment testing). NeuSight's analysis uses the tool's own prediction/label pairs; per-device sample sizes vary (3–18 configurations). **Construct.** Our evaluation prioritizes accuracy; tools may provide value beyond this dimension (e.g., Timeloop's design-space exploration). The supported/partial/unsupported coverage criterion does not capture quality of partial support. **Temporal.** Results reflect tool state as of January 2026; tools under active development may have addressed some limitations, but structural coverage gaps reflect design choices rather than fixable bugs.

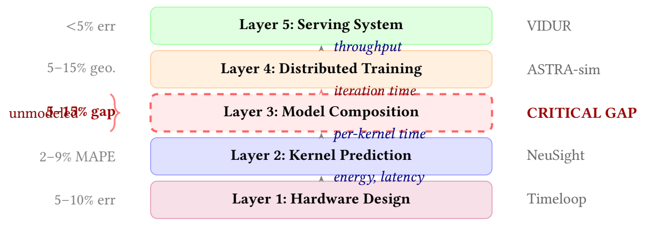


Figure 6: Unified five-layer pipeline. Layer 3 (dashed) is the critical unmodeled gap.

8 Toward a Unified Simulation Pipeline

No single tool spans kernel execution through serving SLAs. Figure 6 shows five layers where 5–9% kernel MAPE is projected to grow to an estimated 10–28% at model level, driven by (i) interface heterogeneity, (ii) calibration mismatch between steady-state models and transient-dominated kernels, and (iii) feedback loops in serving schedulers.

Interface specification. Each pipeline layer must define explicit input/output contracts to enable composition. Layer 1 (Hardware Design) produces an accelerator specification—peak FLOPS, memory bandwidth, cache hierarchy sizes, interconnect topology—that Layer 2 (Kernel Prediction) consumes to build a kernel execution model mapping each operator to a predicted latency and energy cost. The *critical interface gap* lies between Layer 2 and Layer 3: converting a set of per-kernel runtime distributions into a model-level execution graph requires accounting for kernel launch overhead, memory allocation/deallocation between operators, data movement costs, and synchronization barriers—none of which any current kernel-level tool predicts. Layer 3 produces an iteration time distribution that Layer 4 (Distributed Training) consumes alongside a parallelism strategy to produce a job completion distribution, which Layer 5 (Serving System) uses for SLA prediction. Today, each transition requires manual engineering; standardizing these contracts is a prerequisite for any automated pipeline.

Calibration protocol. We propose a two-phase calibration process to manage error accumulation across layers. In the first phase, *single-kernel calibration*, micro-benchmarks for representative kernels (GEMM at various sizes, multi-head attention, layer normalization) are run on target hardware and used to anchor per-kernel predictions from Layer 2. This phase requires access to the target GPU or accelerator for profiling, producing a device-specific correction factor for each kernel class. In the second phase, *composition error correction*, regression models are trained on observed iteration-time residuals—the difference between predicted model-level latency (sum of calibrated kernel times) and measured end-to-end iteration time. This residual captures the composition overhead (launch latency, memory allocation, synchronization) that kernel-level tools miss. Ground truth for this phase requires end-to-end profiling of representative models on target hardware, which is more expensive but can be amortized across workloads sharing similar operator patterns.

Concrete workflow example. Consider predicting LLaMA-70B training throughput on 64 A100-80GB GPUs with 3D parallelism (TP=8, PP=4, DP=2). Layer 1 (Timeloop or an analytical model) provides A100 hardware parameters: 312 TFLOPS FP16, 2 TB/s

HBM bandwidth, 80 GB HBM capacity. Layer 2 (NeuSight) predicts per-kernel latencies for each transformer layer's operators—attention, FFN, normalization—producing a per-layer time estimate. At Layer 3, the composition gap must be bridged: kernel times must be assembled into a full forward-backward iteration accounting for activation checkpointing, pipeline bubble overhead, and micro-batch scheduling. Currently, this step requires manual calculation or custom scripts, introducing 5–15% uncontrolled error. Layer 4 (ASTRA-sim) takes the iteration time and models collective communication (All-Reduce for data parallelism, point-to-point for pipeline parallelism) to produce per-iteration training time including communication overhead. Layer 5 is not needed for training throughput prediction but would be invoked if the question concerned inference serving latency. The chain breaks at Layer 3: no tool automates the kernel-to-model composition, forcing practitioners to either accept large error margins or invest significant engineering effort in manual integration.

Open implementation questions. Realizing this pipeline requires solving several software engineering challenges beyond the modeling research. *Data format standardization* is a prerequisite: tools currently use incompatible input/output formats (YAML for Timeloop, ONNX for NeuSight, Chakra traces [67] for ASTRA-sim, JSON configs for VIDUR), and adopting a common interchange format—potentially extending the Chakra execution trace format—would reduce integration friction. *Calibration dataset curation* is equally critical: the two-phase calibration protocol requires maintained collections of kernel-level and model-level profiling data across hardware generations, which no public dataset currently provides. *Continuous validation infrastructure* must detect when software stack changes (e.g., a new FlashAttention [14] version) invalidate calibrated models, requiring a CI system that periodically re-runs representative benchmarks and flags accuracy regressions. Finally, the pipeline must handle *graceful degradation*: when a tool is unavailable for a particular layer (e.g., no kernel predictor for a novel accelerator), the pipeline should fall back to coarser estimates with explicit uncertainty bounds rather than failing silently.

9 Open Challenges and Future Directions

Six research challenges emerge from our evaluation, each requiring targeted investigation to advance the field from isolated tool development toward a coherent performance prediction ecosystem.

(1) Closing the composition gap. The central research question is whether a learned residual model can predict the composition overhead—kernel launch latency, memory allocation, data movement, synchronization barriers—that causes 2–3% kernel-level error to inflate to 5–12% at model level (Figure 7). Success would be measured by reducing this 5–15% composition error to below 3% on a held-out set of models and hardware configurations. The primary challenge is data sparsity: composition residuals depend on the specific sequence of kernels, the memory state at each transition, and hardware-specific scheduling behavior, making it difficult to collect sufficient training data for a generalizable model. A promising approach is to train lightweight regression models on the residuals between predicted (sum-of-kernels) and measured (end-to-end) iteration times, bootstrapping from profiling data that many organizations already collect for performance debugging.

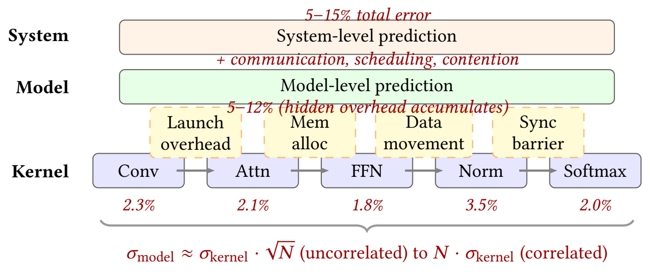

System
5–15% total error
System-level prediction
+ communication, scheduling, contention
Model
Model-level prediction
5–12% (hidden overhead accumulates)
Launch overhead
Mem alloc
Data movement
Sync barrier
Kernel
Conv
Attn
FFN
Norm
Softmax
2.3%
2.1%
1.8%
3.5%
2.0%
$\sigma_{\text{model}} \approx \sigma_{\text{kernel}}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

Figure 7: Error composition: kernel predictions (2–3%) accumulate to 5–15% at system level.

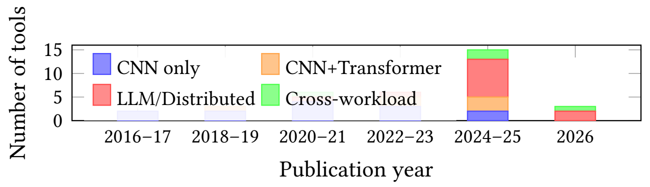


Figure 8: Workload coverage by publication period. MoE and diffusion models remain uncharacterized.

(2) Frontier workload modeling. MoE architectures, diffusion models [35], and dynamic inference techniques (speculative decoding, early exit) introduce execution patterns that no current tool validates against (Figure 8). The key research question is: what kernel primitives and system abstractions must be added to extend tool coverage from the current 16/36 scenarios toward 30/36? MoE models require expert-routing primitives that create load-dependent, input-specific execution paths—a fundamentally different paradigm from the static computation graphs that all current tools assume. Diffusion models introduce iterative denoising loops with varying compute intensity per step, and speculative decoding creates branching execution paths whose cost depends on acceptance rates. As MLPerf evolves to include workloads such as Llama 3.1 405B and FLUX.1, these benchmarks could serve as concrete validation targets for future tool development. The expected challenge is that these *dynamic execution graphs* resist the static trace assumptions underlying tools like ASTRA-sim and VIDUR; new abstractions for conditional and iterative execution are needed.

(3) Hardware transfer. As the accelerator landscape diversifies beyond NVIDIA GPUs to include TPUs [30, 31], AMD GPUs, and processing-in-memory (PIM) architectures [23, 27, 40, 51], the question becomes whether a performance model trained on one architecture family can transfer to another with less than 10% error delta. Success would be demonstrated by training a kernel predictor on GPU profiling data and achieving competitive MAPE on held-out TPU or PIM benchmarks without architecture-specific retraining. The fundamental challenge is architectural heterogeneity: GPU warp scheduling, TPU systolic array dataflow, and PIM near-memory compute have fundamentally different performance bottlenecks, and it is unclear whether any shared representation can capture these differences. NeuSight's cross-vendor results (AMD MI100: 10.80% vs. NVIDIA V100: 5.87%) suggest that even within the

GPU family, architectural differences introduce significant transfer error.

(4) Standardized evaluation. The field lacks a consensus answer to a foundational question: what constitutes simulator correctness? No MLPerf [46, 60, 61] equivalent exists for performance modeling tools, and each tool validates against its own benchmarks using its own metrics, making cross-tool comparison unsound—as our third-party evaluation demonstrates. Success would be a community-maintained CI system that runs a standardized benchmark suite (extending our PerfSim-Survey-2026) against all participating tools and catches accuracy regressions automatically. The primary challenge is the absence of ground truth for novel architectures: validating a simulator requires hardware measurements, and new hardware is often available only to the tool's developers, creating a circular validation problem. Portable trace formats [67] and concurrent surveys [69] are steps toward standardization, but the community has yet to agree on evaluation protocols.

(5) Reproducibility. nn-Meter's complete failure from dependency rot—pickle serialization with an unpinned scikit-learn version rendering the tool non-functional within two years—is not an isolated case but a symptom of a systemic problem. The research question is: what development practices reliably eliminate dependency rot for ML-augmented performance tools? Success would be measured by all evaluated tools passing a *two-year reproducibility test*: given a tool's published artifact and a fresh machine, can a researcher reproduce the claimed results two years after publication? The primary challenge is ML framework churn: PyTorch, TensorFlow, and scikit-learn release breaking changes frequently, and tools that depend on specific internal APIs or serialization formats are vulnerable. Containerization with pinned dependencies (Docker), model serialization in portable formats (ONNX rather than pickle), and automated CI testing against dependency updates are necessary but not yet standard practice in the performance modeling community.

(6) Software stack evolution. Rapidly evolving optimizations such as FlashAttention [14], which reduced attention kernel latency by 2–4× according to the original paper [14], can invalidate performance models trained on prior kernel implementations overnight. The research question is: how quickly do software optimizations invalidate calibrated models, and can this decay rate be predicted? A useful metric is *model half-life*—the time after which a calibrated model's error doubles due to software stack changes. Measuring this requires a longitudinal study tracking model accuracy across successive framework releases, which no group has yet undertaken. The challenge is that such a study requires continuous access to diverse hardware and the engineering capacity to re-profile workloads after each major framework update—a significant infrastructure investment that is difficult to justify for any single research group but could be shared as community infrastructure.

10 Conclusion

We survey 25 ML performance tools, fully evaluate five and assess deployment feasibility of five more, against a 36-scenario benchmark, finding self-reported accuracy unreliable (NeuSight: 2.3% claimed vs. 5.87–27.10%; nn-Meter: no output). The 5–15% composition gap dominates total error; closing it requires validated composition models and community CI.

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

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

[16] Yang Feng, Zhehao Li, Jiacheng Yang, and Yunxin Liu. 2024. LitePred: Transferable and Scalable Latency Prediction for Hardware-Aware Neural Architecture Search. In *Proceedings of the 21st USENIX Symposium on Networked Systems*

Design and Implementation (NSDI). 1–18.

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

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

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

[20] 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.

[21] 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.

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

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

[24] 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

[25] 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.

[26] 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..

[27] 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..

[28] 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).

[29] 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.

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

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

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

[33] 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

[34] 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.

[35] 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.

[36] 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

[37] 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

[38] 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

[39] 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.

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

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

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

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

[44] Ying Li, Yifan Sun, and Adwait Jog. 2023. Path Forward Beyond Simulators: Fast and Accurate GPU Execution Time Prediction for DNN Workloads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14. https://doi.org/10.1145/3613424.3614277 Linear-regression-based DNN execution time predictor achieving 7% error for new DNN prediction and 15.2% for new GPU prediction.

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

[46] 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.

[47] 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.

[48] 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.

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

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

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

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

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

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

[55] Aurick Qiao, Sang Keun Agrawal, Anand Jayarajan, Mostofa 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.

[56] 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.

[57] 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.

[58] 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

[59] 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

[60] 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.

[61] 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.

[62] George F. Riley and Thomas R. Henderson. 2010. The ns-3 Network Simulator. *Modeling and Tools for Network Simulation* (2010), 15–34. https://doi.org/10.1007/978-3-642-12331-3_2

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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).

[68] 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..

[69] Jonas Svedas, Hannah Watson, Nathan Laubeuf, Diksha Moolchandani, Abubakr Nada, Arjun Singh, Dwaipayan Biswas, James Myers, and Debjyoti Bhattacharjee. 2025. A Survey of End-to-End Modeling for Distributed DNN Training: Workloads, Simulators, and TCO. *arXiv preprint arXiv:2506.09275* (2025). Comprehensive survey of distributed DNN training simulators covering workload representation, simulation infrastructure, and TCO/carbon modeling.

[70] 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

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] 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..

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

[77] 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

[78] 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

[79] 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

[80] 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.

[81] 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.

[82] 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

[83] 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.

[84] 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.

[85] 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.

[86] 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.