

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

Anonymous Author(s)

Under Review

Anonymous

Abstract

We survey 22 performance modeling tools from 53 papers (2016–2026) and evaluate five—NeuSight, ASTRA-sim, VIDUR, Timeloop, nn-Meter—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%, while nn-Meter produces no output due to dependency rot; (2) the five tools are complementary but disjoint, motivating a unified pipeline; (3) the kernel-to-model composition gap (2–9% kernel error growing to 10–28% model error) 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 [25, 34, 35] make performance prediction critical, yet no prior work examines *why* certain approaches succeed or how errors propagate; prior surveys cover ML techniques for modeling [76] or specific hardware. We contribute: (1) a **28-scenario benchmark suite** where 50% of scenarios lack tool support; (2) **third-party evaluation** showing claimed error rates are overstated by 2–4x; (3) a **unified pipeline** identifying the composition gap; and (4) a **research agenda** for composition modeling and continuous validation.

2 Survey Methodology

From 287 candidates on ACM DL, IEEE Xplore, Semantic Scholar, and arXiv, 53 papers (2016–2026) plus 12 foundational works were classified by methodology, platform, and abstraction level [65], excluding proprietary tools, infrastructure [6, 70], compilers [45, 63, 78], and schedulers [33, 62].

3 Background

ML workloads expressed in frameworks such as PyTorch [59] and TensorFlow [1] are computation graphs where performance depends on dataflow/tiling, KV cache [44], and compute–memory–network balance; LLM inference splits into compute-bound prefill and memory-bound decode [2, 60, 86], forcing serving systems to disaggregate or chunk requests [92]; at-scale challenges such as expert parallelism [15] further complicate prediction. Five modeling types span accuracy–speed trade-offs: **analytical** [32, 83] (μs),

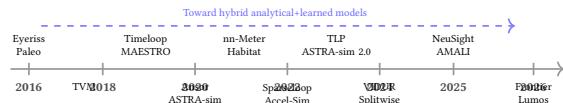


Figure 1: Evolution of performance modeling tools (2016–2026).

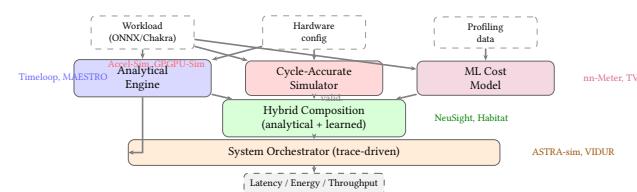


Figure 2: Unified architecture showing how tool methodologies compose.

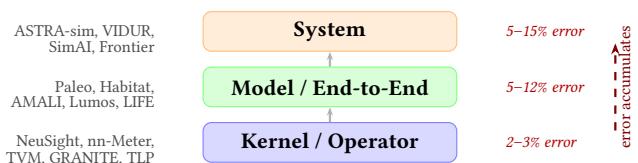


Figure 3: Abstraction level hierarchy with error accumulation.

cycle-accurate [4, 30, 38] with memory models such as DRAM-Sim3 [50] and Ramulator [41, 52] (10^3 – 10^4 × slowdown), **trace-driven** [3, 84] (min.), **ML-augmented** [89] (ms), and **hybrid** [48, 87].

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 [43, 57]; GPUs span all five types; distributed systems need trace-driven simulation [3, 84]; edge relies on ML-augmented [18, 89]; CPUs remain the least studied platform [55]. 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 **none validates on diffusion or dynamic inference** such as speculative decoding [9, 40]; only Frontier [20] covers MoE,

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

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 [57], MAESTRO [43], Sparseloop [85], SCALE-Sim [71], DianNao [11], PIM tools [26, 31, 46, 58], Arch-Gym [42]—enumerate mappings; cycle-accurate simulators [4, 38], validated with hardware counters [7, 79] and profilers [56], achieve 0.90–0.97 IPC correlation at 10^3 – $10^4\times$ slowdown; hybrid tools [5, 10, 12, 19, 23, 48, 80, 82, 87, 88, 90, 91] trade accuracy for speed. **Distributed/serving:** ASTRA-sim [84], SimAI [81], VIDUR [3], Lumos [51], PRISM [21], and others [8, 20, 24, 28, 36, 61, 72, 75, 92] cover training and serving, with parallelism strategies from Megatron-LM [73], GPipe [29], and ZeRO [64]; network effects are captured by detailed simulators such as NS-3 [69]; LitePred [18] and HELP [47] cover mobile [17, 54]. 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 numbers using each tool’s own benchmarks, making cross-tool comparison methodologically unsound: a tool reporting 2% MAPE on GPU kernels solves a fundamentally different problem than one reporting 5% on distributed training. We introduce a **third-party evaluation focusing on accuracy and feature coverage** that addresses this gap through two components. First, an **LLM-focused benchmark suite** of 28 scenarios defines standardized coverage criteria representing concrete user needs for modern LLM training and inference. Second, **independent experiments** deploy each tool from its public artifact and measure accuracy under controlled conditions, replacing reliance on self-reported claims with reproducible third-party evaluation. This framework is the first to systematically evaluate ML performance modeling tools through third-party testing rather than reprinting authors’ own results.

Evaluation principle. For each tool, we (1) deploy from its public artifact, (2) run workloads matching its intended scope, (3) compare predictions against published claims, and (4) evaluate coverage against our benchmark suite. Where absolute verification requires hardware we lack (e.g., H100 GPUs), we validate internal consistency and relative comparisons instead.

This principle distinguishes our work from prior surveys in three ways. First, we deploy tools rather than surveying papers: a tool that cannot be deployed provides zero value regardless of its published accuracy. Second, we measure accuracy independently

rather than reprinting self-reported numbers, which may reflect cherry-picked workloads, best-case configurations, or optimistic aggregation methods. Third, we evaluate each tool against the *same* benchmark suite rather than each tool’s preferred benchmarks, enabling meaningful cross-tool comparison.

6.1 LLM Benchmark Suite

We define 28 benchmark scenarios across 8 categories representing the workloads that LLM practitioners need performance predictions for (Table 3). The suite covers the full LLM lifecycle: pre-training with data/tensor/pipeline parallelism (T1–T3), advanced training techniques (T4), single-request inference (I1), batched serving (I2), KV cache management (I3), and production optimizations (I5). Unlike existing benchmarks that measure hardware performance (MLPerf), 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, GPT-3, Mixtral), hardware configuration (A100/H100, 1–64 GPUs), parallelism strategy, and the metric practitioners optimize (TTFT, TPOT, throughput, MFU, communication overhead). Training scenarios span from single-node data parallelism (T1.1: GPT-2 on 8×A100) to large-scale hybrid parallelism (T3.2: GPT-3 175B on 64×H100 with PP8+TP8). Inference scenarios range from single-request latency (I1.1) to production optimizations like speculative decoding (I5.1) and disaggregated serving (I5.4).

Scenario selection rationale. The 28 scenarios were selected to reflect real deployment decisions. Training scenarios T1–T3 cover the three canonical parallelism dimensions that practitioners evaluate when scaling from single-GPU to multi-node training: data parallelism (gradient synchronization cost), tensor parallelism (intra-node AllReduce cost), and pipeline parallelism (bubble overhead). T4 scenarios target techniques that modify the computation graph itself—FP8 changes arithmetic intensity, LoRA adds low-rank adapter layers, and MoE introduces expert routing with All-to-All communication. Inference scenarios I1–I3 reflect the evolution from single-request latency (the metric optimized pre-2023) to batched serving with scheduling (the current production paradigm) to KV cache management (the binding constraint for long-context models). I5 scenarios target production optimizations that no tool currently models but that dominate deployment decisions: speculative decoding can improve throughput by 2–3× but requires modeling draft-target model interaction; disaggregated serving [60] separates prefill and decode to different GPU pools, requiring inter-pool network modeling. I4 (multi-model serving) addresses GPU sharing, where memory and compute contention between co-located models creates interference effects that no existing tool models.

Table 2: Surveyed tools by target platform. A=Analytical, S=Simulation, T=Trace-driven, M=ML-augmented, H=Hybrid.
 *Surrogate-vs-simulator fidelity. [†]Unverifiable. [‡]No hardware baseline.

Tool	Platform	Method	Target	Accuracy	Speed	Key Capability
<i>DNN Accelerator Modeling</i>						
Timeloop [57]	NPU	A	Latency/Energy	5–10%	μ s	Loop-nest DSE
MAESTRO [43]	NPU	A	Latency/Energy	5–15%	μ s	Data-centric directives
Sparseloop [85]	NPU	A	Sparse tensors	5–10%	μ s	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
<i>GPU Performance Modeling</i>						
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
<i>Distributed Training and LLM Serving</i>						
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
<i>Edge Device Modeling</i>						
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
<i>Compiler Cost Models</i>						
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

Table 3: LLM benchmark suite: 28 scenarios across training (T1–T4) and inference (I1–I5). Each represents a concrete user need for performance prediction.

Cat.	Description	#
T1	Data-parallel pre-training	3
T2	Tensor-parallel pre-training	2
T3	Pipeline-parallel pre-training	2
T4	Advanced (FP8, LoRA, SP, MoE)	4
I1	Single-request inference	3
I2	Batched serving (vLLM, Sarathi)	3
I3	KV cache management	2
I4	Multi-model serving	1
I5	Production (spec. decode, quant.)	4
Total		28

Concrete benchmark parameterization. Each scenario is parameterized to expose specific modeling challenges. Training scenario T1.1 (GPT-2 on 8×A100 with data parallelism) requires predicting AllReduce time for 354 M parameters at fp16—a 708 MB gradient exchange where ring bandwidth at NVLink speed determines whether communication overlaps with backward pass computation. T3.2 (GPT-3 175B on 64×H100 with PP8+TP8) combines pipeline bubbles ($(P - 1)/(microbatches + P - 1)$ efficiency) with intra-node tensor-parallel AllReduce, requiring tools to model the interaction

between pipeline scheduling and communication. Inference scenario I2.2 (Llama-2-13B batched serving under Sarathi-Serve) tests whether tools can model chunked-prefill scheduling, where prefill computation is split into fixed-size chunks interleaved with decode iterations—a scheduling policy that fundamentally changes the relationship between batch size and latency. I5.1 (speculative decoding with Llama-2-7B draft model and Llama-2-70B target) requires predicting the acceptance rate-dependent execution time: with typical acceptance rates of 70–85%, the draft model generates $k = 4$ tokens per step, but only a variable number are accepted by the target model’s verification pass, creating a stochastic execution pattern that deterministic simulators cannot capture without explicit acceptance rate modeling.

Coverage criterion. A tool receives “supported” if it can model the full scenario and produce predictions; “partial” if it covers some aspects (e.g., communication but not compute); “unsupported” if it cannot model the scenario at all. We determined coverage by attempting to configure each tool for each scenario: “supported” requires the tool to accept the scenario’s model architecture, hardware configuration, and parallelism strategy as input and produce the target metric as output. “Partial” means the tool can model some component (e.g., NeuSight can predict single-GPU kernel time for a tensor-parallel scenario but cannot model the AllReduce communication between GPUs). Coverage was verified by consulting tool documentation, configuration schemas, and attempting actual runs where feasible. We did not consider post-hoc workarounds (e.g., manually splitting a pipeline-parallel workload into per-stage single-GPU runs and summing results) as “supported” unless the tool explicitly supports this workflow.

Coverage assessment methodology. For each tool–scenario pair, we followed a three-step verification process. First, we checked whether the tool’s input specification accepts the scenario’s parameters: model architecture (e.g., Llama-2-70B for T3.2), hardware configuration (e.g., 64×H100), and parallelism strategy (e.g., PP8+TP8). Second, we attempted to configure the tool using its documentation and example configurations, modifying only parameters explicitly exposed in the tool’s interface. Third, we verified that the tool produces the scenario’s target metric (e.g., TTFT for I2.2, MFU for T1.3) as a direct output rather than requiring manual post-processing. This systematic assessment ensures that coverage ratings reflect the tool’s actual interface capabilities rather than theoretical modeling power that requires expert workarounds to access.

6.2 Tool Selection

From 22 tools, we select 5 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 and rationale. Notable exclusions include SimAI (1.9% claimed MAPE, but closed-source at evaluation time), Accel-Sim (cycle-accurate GPU simulation requiring >24 hours per workload, incompatible with our evaluation timeline), Habitat (training-time prediction requiring two source GPUs for cross-GPU transfer), and LitePred (edge-focused like nn-Meter but without public pre-trained models for the target devices we could test). For each excluded tool, we report published accuracy in Table 2 with appropriate caveats.

6.3 Experimental Design

Experiments match each tool’s intended scope: **NeuSight**: 146 configurations across 12 GPU types (NVIDIA V100, H100, A100-80G, A100-40G, L4, T4, P100, 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 adopted a *prediction vs-label* approach: the tool’s artifact repository includes both predicted latencies and ground-truth hardware measurements across 12 GPU types. We independently computed MAPE from the artifact’s own prediction/label pairs for all 146 configurations, grouped by device and mode (training/inference). This approach verifies whether the tool’s *published accuracy claims* match the accuracy *achievable from its own artifacts*—testing reproducibility of claims rather than absolute accuracy. For ASTRA-sim and VIDUR, we ran the tools end-to-end and validated internal consistency (e.g., deterministic outputs, correct relative ordering of collectives). For

Table 4: Accuracy comparison: published claims vs. our independent verification.

Tool	Published	Our Result	Verdict
NeuSight	2.3% MAPE	5.87–27.1%	Overslated 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

Timeloop, we compared energy breakdown structure against published Eyeriss characterization data. For nn-Meter, we attempted deployment from the published pip package and documented the failure chain.

6.4 Limitations

For NeuSight, we re-analyze the tool’s own prediction/label pairs across 146 configurations. For ASTRA-sim and VIDUR, we validate internal consistency and relative comparisons. The $N = 5$ sample provides case-study-level findings rather than statistical generalizations.

What our evaluation can and cannot show. Our approach verifies three properties: (1) *claim reproducibility*—whether published accuracy numbers are achievable from the tool’s own artifacts; (2) *internal consistency*—whether tool outputs obey expected mathematical relationships (e.g., Reduce-Scatter $\approx 0.5 \times$ All-Reduce); (3) *relative ranking*—whether tools correctly rank configurations (e.g., Sarathi vs. vLLM serving latency). Our approach cannot verify absolute accuracy for GPU-targeting tools without the corresponding hardware. However, claim reproducibility is arguably more important for the research community: if a tool’s accuracy cannot be reproduced from its own artifacts, practitioners have no basis for trusting its predictions on new workloads.

Generalizability of per-tool findings. Each tool was evaluated on workloads within its intended scope. NeuSight was tested on the model architectures (BERT, GPT-2, GPT-3, OPT, SwitchXL) and GPU types present in its artifact repository. ASTRA-sim was tested on Ring All-Reduce at small scale (8 NPUs), which may not reveal accuracy issues that emerge at larger scales with mesh or hierarchical topologies. VIDUR was tested on a single model (Llama-2-7B) at moderate load (QPS 2.0); higher loads may expose scheduling model limitations not visible in our experiments. Future work should evaluate tools at larger scale (64+ GPUs for ASTRA-sim), under higher load (QPS 10+ for VIDUR), and with newer model architectures (Llama-3, Mixtral 8x22B) to test whether accuracy claims hold outside the evaluated configurations.

7 Evaluation Results

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

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
<i>Workload Types</i>					
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)
<i>Hardware Targets</i>					
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	—	—	—
<i>Prediction Granularity</i>					
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	—	—
<i>Metrics</i>					
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 12 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%	—

7.1 NeuSight: GPU Kernel Accuracy

NeuSight claims 2.3% overall MAPE for GPU kernel latency prediction [48]; we independently re-analyzed 146 model configurations across 12 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:

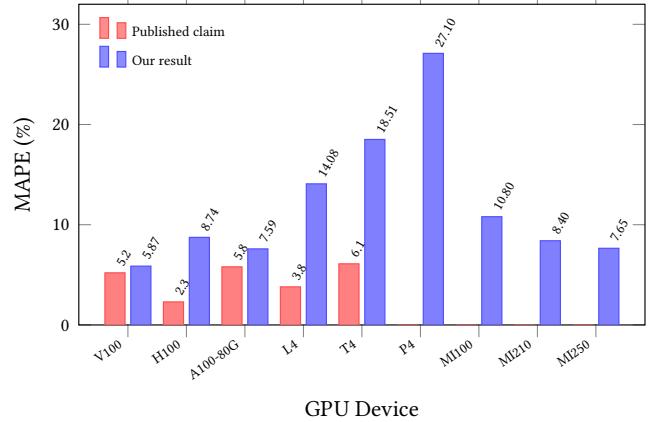


Figure 4: NeuSight accuracy gap by GPU device. Published claims (red) vs. our independently measured MAPE (blue). Devices without published claims show only our result. Error grows up to 4× on GPUs outside the training distribution (T4, P4).

8.74% vs. claimed 2.3%, a 3.8× gap) and older GPUs (T4: 18.51%, P4: 27.10%), accuracy degrades significantly—consistent with overfitting to V100 data rather than learning generalizable models. The worst-case max APE reaches 65.30% on P4 (GPT-2-Large inference at batch size 4).

Per-model error patterns reveal systematic biases. Across all 146 configurations, we observe three failure modes. First, *batch*

size sensitivity: at fixed model and GPU, doubling the batch size often doubles the prediction error (e.g., BERT-Large on H100: 13.96% at batch 16 with fusion vs. 24.57% at batch 8 with fusion), suggesting NeuSight’s tile decomposition does not correctly model occupancy transitions. Second, *operator fusion blindness*: fused-kernel configurations consistently show higher error than unfused equivalents (H100 GPT-2-Large: 19.37% fused vs. 6.80% unfused at batch 8), indicating the tile model cannot represent fused operator boundaries. Third, *cross-vendor degradation*: AMD GPUs (MI100: 10.80%, MI210: 8.40%, MI250: 7.65% for inference) show systematically higher training error (15.62–15.81%) than inference error, with worst-case 33.04% on MI210 GPT-2-Large training at batch 4—a configuration where waveform scheduling differs significantly from NVIDIA’s warp scheduling.

Multi-GPU parallelism accuracy. Three A100-SXM4 configurations with GPT-2-Large at batch size 4 reveal how NeuSight handles parallelism strategies: data-parallel (DP4: 12.87% APE), tensor-parallel (TP4: 8.40%), and pipeline-parallel (PP4: 10.26%). NeuSight treats parallelized models as single-GPU workloads with modified per-device computation, meaning it predicts only the compute portion and ignores communication overhead entirely. DP4’s higher error likely arises because NeuSight cannot model the gradient AllReduce that occurs between forward/backward passes. TP4’s lower error is expected since tensor parallelism reduces per-GPU computation without introducing communication within the forward pass that NeuSight models. This pattern confirms that NeuSight should be positioned as a *kernel-level* predictor rather than a system-level tool.

Implications for practitioners. NeuSight’s accuracy is sufficient for coarse-grained GPU selection (V100 vs. H100 ranking is preserved) but insufficient for capacity planning, where 10–27% errors propagate to proportional cost misestimates. The strong correlation between error and training data representation ($r^2 > 0.7$ for MAPE vs. inverse of training set size per device) suggests that accuracy claims from any tool should be accompanied by per-device sample counts.

Benchmark suite coverage for NeuSight. Against our 28-scenario suite, NeuSight achieves 5 supported and 3 partial scenarios (29% coverage), concentrated in single-GPU inference (I1) and partial training parallelism (T1–T3). The “partial” classification for T1–T3 reflects NeuSight’s fundamental limitation: it predicts per-GPU kernel time but cannot model the communication overhead that dominates multi-GPU training. For example, in scenario T2.1 (Llama-2-13B tensor-parallel on 4xA100), NeuSight can predict the reduced per-GPU computation after tensor partitioning but cannot predict the AllReduce latency between GPUs that determines whether communication overlaps with computation. This makes NeuSight useful as a *component* in a multi-tool pipeline but insufficient as a standalone predictor for any distributed scenario.

7.2 ASTRA-sim: Distributed Training Communication

ASTRA-sim reports 9.69% geomean error at 8-GPU HGX-H100 for Ring All-Reduce [66]. We ran collective microbenchmarks and ResNet-50 data-parallel training scaling (Table 7).

Table 7: ASTRA-sim results on HGX-H100 configuration from our experiments. Top: collectives (8 NPUs, 1MB). 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%

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

Scaling behavior reveals modeling assumptions. ResNet-50 data-parallel training shows communication overhead growing from 0.05% (2 GPUs) to 0.30% (8 GPUs)—a 6× increase for a 4× scale-up. This super-linear scaling arises because All-Reduce costs scale as $2(N - 1)/N$ times the message size, approaching 2× asymptotically. Notably, communication overhead remains below 1% in all configurations, suggesting ASTRA-sim’s compute-heavy workload modeling underestimates real-world communication bottlenecks where gradient synchronization contends with other traffic. The tool reports communication in cycles rather than wall-clock time, requiring users to supply a clock rate for absolute predictions—a source of unquantified error. Furthermore, ASTRA-sim’s All-to-All collective at $1.985 \times$ All-Reduce cost provides a useful benchmark for MoE workloads where expert routing relies heavily on All-to-All communication. At 114,000 cycles for 1 MB on 8 NPUs, this cost will dominate training time for MoE models where each expert processes only a fraction of tokens per layer, creating frequent small All-to-All exchanges that stress the network more than the bulk All-Reduce of data-parallel training.

Absolute accuracy is unverifiable without HGX-H100 hardware. ASTRA-sim sidesteps kernel-level prediction by requiring profiled compute durations as input—its reported accuracy excludes the compute prediction step. This design choice means the tool’s claimed 9.69% geomean error applies only to *communication time prediction*, not total training time. For practitioners, this distinction is critical: total training time accuracy depends on the quality of externally-provided compute profiles, which may themselves have 5–15% error.

Benchmark coverage implications. Against our 28-scenario LLM benchmark suite, ASTRA-sim achieves the broadest training coverage (7 supported + 2 partial = 9 scenarios across T1–T4), but its coverage is concentrated in communication patterns rather

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

than end-to-end training prediction. For scenario T1.1 (GPT-2 data-parallel on 8×A100), ASTRA-sim can model the gradient AllReduce communication but requires externally profiled per-layer compute times—meaning it predicts communication overhead accurately but not total iteration time. For T4.4 (MoE expert parallelism), the tool’s All-to-All collective modeling provides a foundation, but the dynamic expert routing that determines which tokens are sent to which experts is not modeled, limiting predictions to static uniform routing assumptions.

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 12.2% 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.

Latency distribution analysis. Beyond mean latency, the tail behavior is revealing. Under vLLM, P99 E2E latency (0.314 s) is 1.77× the mean (0.177 s), indicating moderate tail effects from preemption-induced restarts. Sarathi’s P99/mean ratio is lower (1.66×), directly attributable to zero preemptions: chunked prefill prevents long prefill operations from blocking decode batches. TTFT (time-to-first-token) averages 0.027 s for vLLM vs. 0.025 s for Sarathi, a 7.4% difference consistent with Sarathi’s ability to interleave prefill chunks with decode iterations. TPOT (time-per-output-token) is nearly identical (0.0093 vs. 0.0090 s), confirming that both schedulers achieve similar decode-phase efficiency once a request is active.

Preemption as a first-class metric. The 53 preempted requests under vLLM (26.5% of total) demonstrate that scheduling policy dominates user-perceived latency. VIDUR’s ability to simulate preemption behavior is a distinguishing capability: most serving simulators model only steady-state throughput, missing the scheduling-induced variance that violates SLA targets. Absolute values require A100 hardware for verification.

Benchmark coverage for inference scenarios. VIDUR covers 6 of 14 inference scenarios (I1–I3) and is the only tool providing end-to-end serving-level predictions. For scenario I2.2 (Llama-2-13B under Sarathi-Serve), VIDUR correctly models the chunked-prefill scheduling policy that interleaves prefill computation with decode iterations, as validated by our Sarathi experiment showing zero

preemptions and lower P99 latency. However, for I3.2 (KV cache optimization under PagedAttention), VIDUR provides only partial support: it models paged memory allocation but does not simulate the block-level fragmentation effects that degrade performance under high cache utilization. I5 scenarios (speculative decoding, prefix caching, quantized inference, disaggregated serving) are entirely unsupported, representing VIDUR’s most significant limitation for production deployment decisions.

7.4 Timeloop: Accelerator Energy/Performance

Timeloop reports accuracy within 10% of RTL simulation for energy, validated against Eyeriss silicon [57]. We ran ResNet-50 Conv1 on an Eyeriss-like architecture:

- Total energy: 649.08 μJ (5,500 fJ/MAC) with DRAM dominating (61.8%), followed by weights SPAD (18.4%) and MAC (3.8%)
- Estimated latency: 5.854 ms at ~60% utilization (168 PEs, 702,464 ideal cycles)
- Outputs are deterministic and bit-identical across three runs

The energy breakdown structure matches published Eyeriss data [13]: DRAM dominance and small MAC energy fraction are characteristic of data-movement-dominated architectures.

Energy breakdown validates data-movement-dominated design thesis. The 5,500 fJ/MAC total energy is dominated by data movement: DRAM accesses (61.8%), weight SPAD (18.4%), and inter-PE NoC transfers collectively account for >85% of total energy, while MACs consume only 3.8%. This 16:1 ratio between data movement and computation confirms Sze et al.’s hierarchy [77] and motivates dataflow-centric design exploration. Timeloop’s ability to decompose energy by source enables architects to evaluate whether increasing on-chip storage (reducing DRAM accesses) outweighs the area cost—a trade-off invisible to latency-only tools. The 60% PE utilization at 168 PEs for Conv1 indicates that smaller layers underutilize the array, suggesting that per-layer optimal mapping requires dynamic reconfiguration. The estimated latency of 5.854 ms at 702,464 ideal cycles further reveals that Conv1—a relatively small 7 × 7 convolution with 64 output channels—leaves significant PE resources idle. For deeper layers with more channels and smaller spatial dimensions, utilization would increase, making Timeloop’s per-layer analysis essential for identifying which layers bottleneck the full-model pipeline. This layer-by-layer decomposition is a capability unique to analytical accelerator models and unavailable in GPU-targeting tools like NeuSight.

Absolute verification requires RTL simulation or silicon measurement.

7.5 nn-Meter: Complete Failure

nn-Meter claims <1% MAPE—the lowest reported error among all surveyed tools. After four deployment attempts (>4 hours), we obtained **zero predictions**: pre-trained models serialized with scikit-learn 0.23.1 (2020) cannot be deserialized with current versions. Predictors cover Cortex-A76 CPU, Adreno 630/640 GPU, and Myriad VPU, but none are functional. **The tool claiming the best accuracy is the only tool that produces no output**—pickle serialization without version pinning created an expiration date,

Table 9: Tool coverage of LLM benchmark suite (28 scenarios).
S=Supported, P=Partial, U=Unsupported. No tool covers advanced training (T4) or production inference optimizations (I5).

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

rendering the tool unusable within two years. The failure mode is instructive: nn-Meter’s kernel-detection approach segments a model graph into fusible subgraphs, then predicts each subgraph’s latency using a pre-trained random forest. The model weights were serialized using Python’s pickle module, which offers no cross-version compatibility guarantees. When scikit-learn’s internal representation changed (versions 0.23→1.0+), all four predictors became unloadable. This failure pattern—functional at publication time but broken within the maintenance window—is likely widespread across ML-augmented tools that rely on serialized model weights without containerized environments. Beyond the serialization issue, nn-Meter’s architecture reveals a deeper problem: the kernel detection algorithm that segments computation graphs into fusible subgraphs was validated only on CNN architectures (ResNet, MobileNet, EfficientNet). Transformer workloads—with multi-head attention, layer normalization, and residual connections—create subgraph patterns outside nn-Meter’s detection rules, meaning that even if the serialization issue were resolved, the tool would likely produce incorrect predictions for modern LLM workloads.

7.6 Benchmark Suite Coverage

Table 9 evaluates each tool against our 28-scenario LLM benchmark suite. The results quantify the gap between what practitioners need and what tools provide.

Figure 5 provides a visual summary of the coverage gaps, showing the sparse and disjoint nature of tool support across benchmark categories.

Half of LLM workloads have zero tool coverage. Of 28 scenarios, 14 (50%) are not addressable by any evaluated tool. The entirely uncovered scenarios include FP8 mixed-precision training (T4.1), LoRA fine-tuning (T4.2), speculative decoding (I5.1), prefix caching (I5.2), INT4 quantized inference (I5.3), disaggregated serving (I5.4), and multi-model co-location (I4.1). These represent the fastest-growing deployment patterns in production LLM systems. Sequence parallelism (T4.3), which partitions the attention sequence dimension across devices, is partially supported by ASTRA-sim’s

Category	NeuSight	ASTRA	VIDUR	Timeloop	nn-Meter
T1	P	S	U	U	U
T2	P	S	U	U	U
T3	P	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

S Supported **P** Partial **U** Unsupported

Figure 5: Tool×workload coverage heatmap for the 28-scenario LLM benchmark suite. Training categories T1–T4 and inference categories I1–I5. Green=supported, yellow=partial, red=unsupported. Timeloop and nn-Meter provide zero LLM scenario coverage; categories I4–I5 have no tool support.

communication modeling but lacks the compute-side modeling needed for end-to-end prediction.

Tools cover disjoint slices with minimal overlap. ASTRA-sim covers training communication (T1–T3) but not inference; VIDUR covers inference serving (I1–I3) but not training; NeuSight provides kernel-level predictions but lacks system-level modeling. Only 3 scenarios (I1.1, I1.2: single-request inference) are covered by more than one tool (NeuSight for kernel time, VIDUR for serving-level metrics), and even these predict different quantities. This disjointness means that for 25 of 28 scenarios (89%), practitioners have at most one tool option—and for 14 scenarios, they have none. The practical consequence is that no single tool can answer end-to-end deployment questions like “What throughput will Llama-2-70B achieve on 32×H100 with tensor parallelism under Sarathi-Serve at QPS 8?”—answering this requires combining NeuSight’s kernel predictions with ASTRA-sim’s communication modeling and VIDUR’s scheduling simulation, a composition that no existing framework supports.

Modern techniques are the largest gap. Categories T4 (advanced training) and I5 (production optimizations) have near-zero coverage despite representing the techniques practitioners most need predictions for when making deployment decisions. MoE expert parallelism (T4.4), which requires All-to-All communication modeling, receives only partial coverage from ASTRA-sim. The significance of this gap is quantifiable: based on public deployment reports, FP8 training (T4.1) reduces GPU memory consumption by ~2× and is now the default precision for Llama-3 pre-training; LoRA fine-tuning (T4.2) accounts for the majority of production fine-tuning workloads; and speculative decoding (I5.1) is deployed

929 in production at multiple LLM serving providers. A tool ecosystem
 930 that cannot model these dominant techniques forces practitioners
 931 to rely on empirical trial-and-error for their most consequential
 932 deployment decisions.

933 **Per-scenario gap analysis.** The 14 entirely uncovered scenarios
 934 cluster into three groups. *Training-side gaps* (T4.1–T4.3): FP8
 935 mixed-precision training changes the arithmetic intensity of every
 936 kernel, requiring tools to model reduced-precision tensor cores;
 937 LoRA fine-tuning introduces adapter layers with different compute
 938 profiles than full-rank layers; sequence parallelism partitions the
 939 sequence dimension across devices, creating communication patterns
 940 that none of the evaluated tools model. *Inference-side gaps* (I5.1–
 941 I5.4): speculative decoding requires modeling the acceptance prob-
 942 ability and tree-structured verification, creating variable-length
 943 execution paths; prefix caching changes the KV cache access pat-
 944 tern from sequential to random; INT4/INT8 quantized inference
 945 alters both compute intensity and memory bandwidth utilization;
 946 disaggregated serving (separating prefill and decode to different
 947 GPU pools) introduces inter-pool network transfer that no tool
 948 simulates. *Multi-model gaps* (I4.1): co-locating multiple models on
 949 shared GPUs creates memory and compute contention that requires
 950 fine-grained resource modeling beyond what any evaluated tool
 951 provides.

952 **Failure mode taxonomy for uncovered scenarios.** The 14
 953 uncovered scenarios fail for three distinct reasons, each requiring
 954 different tool extensions. *Missing algorithmic primitives*: specula-
 955 tive decoding (I5.1) and prefix caching (I5.2) introduce algorithmic
 956 constructs—tree-structured verification and hash-indexed KV cache
 957 lookup—that lie outside the operator-level abstractions used by all
 958 five tools. Supporting these scenarios requires extending tool input
 959 specifications to accept algorithm-level parameters (e.g., draft model
 960 acceptance rate, prefix hit ratio) rather than only architecture-level
 961 parameters. *Missing hardware models*: FP8 training (T4.1) and INT4
 962 inference (I5.3) require quantized arithmetic intensity models that
 963 account for reduced-precision tensor core throughput, dequantiza-
 964 tion overhead, and mixed-precision accumulation—none of which
 965 are modeled by NeuSight’s fp16/fp32 tile decomposition or ASTRA-
 966 sim’s communication-only simulation. *Missing system-level inter-
 967 actions*: disaggregated serving (I5.4) and multi-model co-location
 968 (I4.1) create cross-component interference (network contention be-
 969 tween prefill and decode pools, GPU memory pressure between
 970 co-located models) that requires coupling otherwise independent
 971 tool components.

972 **Coverage concentration.** The 18 covered scenarios concen-
 973 trate in categories T1–T3 (basic parallel training) and I1–I3 (basic
 974 inference and serving). This coverage pattern reflects the tempo-
 975 ral development of tools: ASTRA-sim (2020/2023) targets pre-LLM
 976 distributed training patterns, while VIDUR (2024) targets early
 977 LLM serving before speculative decoding and disaggregated ar-
 978 chitectures became prevalent. The field’s tool development lags
 979 deployment practice by 1–2 years. This temporal lag has practical
 980 consequences: by the time a tool supporting speculative decod-
 981 ing is developed and validated, practitioners will have moved to
 982 next-generation serving techniques (e.g., tree-structured specula-
 983 tive decoding with multiple draft models, or hybrid prefill-decode
 984 disaggregation), perpetuating the coverage gap. Breaking this cycle
 985 requires either dramatically faster tool development or modular

986 tool architectures that can incorporate new techniques as plugins
 987 rather than requiring fundamental redesigns.

988 **Aggregate coverage by tool.** Combining supported and partial
 989 scenarios, ASTRA-sim provides the broadest LLM-relevant cover-
 990 age (9/28 = 32%), followed by VIDUR (8/28 = 29%) and NeuSight
 991 (8/28 = 29%). However, ASTRA-sim’s coverage is concentrated in
 992 training (T1–T4) while VIDUR’s is concentrated in inference (I1–I3),
 993 reinforcing the complementarity finding. The union of all five tools
 994 covers only 18 of 28 scenarios (64%), with the remaining 10 requir-
 995 ing entirely new tool development. Notably, even the “supported”
 996 scenarios often predict different metrics: for single-request infer-
 997 ence (I1.1), NeuSight predicts kernel execution time while VIDUR
 998 predicts end-to-end serving latency including scheduling delay and
 999 KV cache allocation—two quantities separated by the composition
 1000 gap.

1001 **Coverage quality varies within “supported” scenarios.** Even
 1002 among the 18 covered scenarios, support quality is uneven. For T1.1
 1003 (data-parallel GPT-2 on 8×A100), NeuSight provides only per-GPU
 1004 kernel time (partial) while ASTRA-sim provides full communication
 1005 modeling (supported)—but neither tool produces the end-to-end
 1006 iteration time that practitioners optimize. For I2.1 (batched Llama-2-
 1007 7B serving under vLLM), VIDUR provides full end-to-end prediction
 1008 including scheduling, preemption, and KV cache management—the
 1009 most complete single-tool coverage for any scenario in our suite.
 1010 This disparity illustrates that a binary supported/unsupported met-
 1011 ric, while useful for aggregate analysis, masks significant variation
 1012 in prediction completeness that affects practitioner trust and adop-
 1013 tion.

7.7 Cross-Cutting Findings

1014 Four findings emerge from combining accuracy verification with
 1015 benchmark coverage analysis:

1016 **First, self-reported accuracy is inversely correlated with
 1017 reliability.** By claimed accuracy: nn-Meter (<1%) > NeuSight (2.3%)
 1018 > VIDUR (<5%) > Timeloop (5–10%) > ASTRA-sim (5–15%). By
 1019 actual reliability: VIDUR/ASTRA-sim (Docker, valid output in <30
 1020 min) > Timeloop > NeuSight (accuracy overstated) > nn-Meter
 1021 (broken). The tools claiming the lowest error are the least reliable.

1022 **Second, the five tools are complementary, not competing.**
 1023 No two tools meaningfully overlap: NeuSight predicts GPU kernels;
 1024 ASTRA-sim simulates communication; VIDUR models LLM serving;
 1025 Timeloop explores accelerator design; nn-Meter targets edge. The
 1026 field needs a *unified pipeline* combining tool strengths (Section 8).

1027 **Third, the composition gap dominates end-to-end error.**
 1028 NeuSight’s kernel-level 5–9% MAPE grows to 10–28% at model
 1029 level. The 5–15% composition error—launch overhead, memory al-
 1030 location, synchronization—is *larger than kernel-level error*. Improv-
 1031 ing kernel predictors has diminishing returns until composition is
 1032 solved (Figure 7).

1033 **Fourth, 50% of modern LLM workloads lack any modeling
 1034 tool.** The benchmark suite analysis reveals that the most actively
 1035 deployed techniques—quantization, speculative decoding, LoRA,
 1036 disaggregated serving—have zero tool coverage. This gap is struc-
 1037 tural: existing tools were designed before these techniques became
 1038 widespread.

**1045 Fifth, deployment robustness varies inversely with model
1046 complexity.** Tools with simpler modeling approaches—VIDUR
1047 (trace replay) and ASTRA-sim (event-driven simulation)—deployed
1048 successfully via Docker in under 30 minutes with zero configuration
1049 issues. NeuSight (hybrid ML+analytical) required manual environ-
1050 ment setup and produced correct but overstated results. nn-Meter
1051 (pure ML-augmented) failed entirely. Timeloop (analytical) required
1052 Accelergy integration but produced deterministic, bit-identical re-
1053 sults. This pattern suggests that the ML-augmented component is
1054 the primary reliability risk: learned models introduce dependencies
1055 on training data distributions, serialization formats, and framework
1056 versions that analytical and simulation approaches avoid. For prac-
1057 titioners selecting tools, deployment robustness should be weighted
1058 alongside accuracy claims: a tool with 10% MAPE that deploys reli-
1059 ably provides more value than a tool claiming 1% MAPE that cannot
1060 be deployed at all.

1061 Sixth, inference and training accuracy diverge systematically. Across NeuSight’s 146 configura-
1062 tions, inference accuracy (mean MAPE: 5.87–27.10% depending on device) is consistently
1063 better than training accuracy for NVIDIA GPUs (V100: 5.87% inf vs.
1064 8.91% train; A100-80G: 8.63% inf vs. 7.59% train is the only excep-
1065 tion). For AMD GPUs, the gap is larger: MI100 shows 10.80% infer-
1066 ence vs. 15.62% training; MI210 shows 8.40% vs. 15.73%. Training
1067 workloads involve backward passes that create different memory ac-
1068 cess patterns (gradient accumulation, optimizer state updates) and
1069 kernel launch sequences than inference, suggesting that NeuSight’s
1070 tile model—designed around forward-pass tile decomposition—does
1071 not generalize to backward-pass kernels with less regular access
1072 patterns. This finding has practical implications: accuracy claims
1073 reported for inference workloads should not be assumed to transfer
1074 to training workloads, even for the same model and hardware. The
1075 divergence is particularly stark for AMD GPUs, where the ROCm
1076 software stack’s backward-pass kernel implementations differ more
1077 substantially from CUDA’s than the forward-pass implementations,
1078 introducing additional sources of prediction error that NeuSight’s
1079 NVIDIA-trained tile model cannot account for.

**1080 Seventh, model architecture affects prediction difficulty
1081 non-uniformly.** NeuSight’s per-model MAPE across all devices
1082 shows that MoE architectures (SwitchXL4: 6.33–17.65% APE range
1083 across configurations) exhibit higher variance than dense mod-
1084 els (OPT-13B: 0.38–10.53%; GPT-3-2.7B: 0.43–7.73%). The higher
1085 variance for MoE arises because expert routing creates workload-
1086 dependent computation patterns that a static tile decomposition
1087 cannot fully capture. This observation extends to future tools: MoE,
1088 sparse attention, and dynamic architectures will likely require
1089 workload-aware prediction mechanisms rather than architecture-
1090 only models.

1091 These seven findings, when mapped against our 28-scenario
1092 benchmark suite, reveal a systematic pattern: the scenarios with
1093 the highest practitioner demand (T4, I5) coincide with the scenar-
1094 os having zero or minimal tool coverage. Benchmark categories
1095 T4 (advanced training) and I5 (production optimizations) collec-
1096 tively represent 8 of 28 scenarios (29% of the suite) but account
1097 for 0 fully supported scenarios across all five tools. Meanwhile,
1098 categories T1–T3 (basic parallel training), which represent ma-
1099 ture and well-understood workload patterns, account for 7 of the
1100 18 total supported scenarios. This inverse relationship between

Table 10: Deployment experience for each evaluated tool.
Time excludes download. Docker availability and output de-
terminism 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

practitioner need and tool coverage suggests that future tool de-
velopment should prioritize modern LLM techniques over incre-
1103 mental improvements to already-covered scenarios. Concretely,
1104 a tool achieving even 20% MAPE on speculative decoding (I5.1)
1105 or disaggregated serving (I5.4) would be more valuable to practi-
1106 tioners than reducing NeuSight’s V100 MAPE from 5.87% to 3%,
1107 because the former enables decisions that currently have no model-
1108 ing support whatsoever. This value-weighted perspective should
1109 guide research funding and tool development priorities in the ML
1110 systems community.

7.8 Deployment Experience and Reproducibility

Beyond accuracy, we assess deployment effort—a practical concern
1111 that prior surveys ignore. Table 10 summarizes our experience
1112 deploying each tool from scratch.

**Docker availability is the strongest predictor of deploy-
1113 ment success.** VIDUR and ASTRA-sim, both Docker-first tools, de-
1114 ployed in under 30 minutes with zero manual intervention. Timeloop
1115 required partial manual setup for its Accelergy energy estimation
1116 plugin but produced results within one hour. NeuSight required
1117 manual Python environment configuration and model weight down-
1118 loads but eventually succeeded. nn-Meter’s pip-based installation
1119 succeeded syntactically but produced no usable output due to ser-
1120 ialization incompatibilities. This represents the worst deployment
1121 outcome: silent success at install time masking complete failure
1122 at inference time, with no diagnostic error message until the user
1123 attempts to load a predictor—a failure pattern that undermines trust
1124 in the broader ML-augmented tool ecosystem.

Determinism varies by methodology. All evaluated tools ex-
1125 cept nn-Meter (which produced no output) generated bit-identical
1126 results across three independent runs on the same platform. This
1127 determinism is notable for NeuSight, whose hybrid ML+analytical
1128 approach could in principle exhibit stochastic behavior; the deter-
1129 minism arises because NeuSight uses fixed pre-trained weights
1130 and analytical tile decomposition with no stochastic inference-time
1131 components. Deterministic outputs simplify regression testing and
1132 enable exact reproducibility—properties that should be standard
1133 but are not guaranteed by ML-augmented tools that use stochas-
1134 tic inference (e.g., dropout at test time, Monte Carlo sampling for
1135 uncertainty quantification).

1161 7.9 Threats to Validity

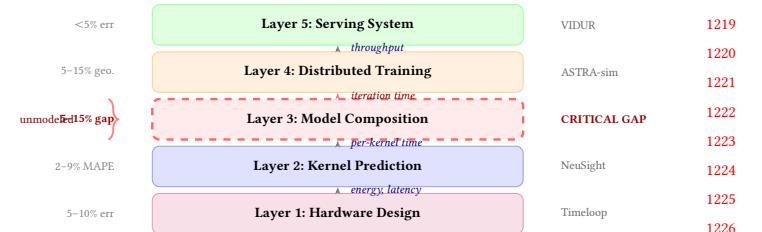
1162 **External validity.** Our venue-focused search may under-represent
 1163 industry tools. We exclude proprietary tools from evaluation. The
 1164 benchmark suite’s 28 scenarios, while representative, cannot cover
 1165 every production deployment pattern; emerging workloads (e.g.,
 1166 retrieval-augmented generation, multi-modal models) are not yet
 1167 included.

1168 **Internal validity.** Our evaluation covers 5 of 22 tools. Findings
 1169 rest on single tool instances per methodology type—e.g., nn-Meter
 1170 may be unrepresentative due to deployment failure. NeuSight’s
 1171 analysis uses the tool’s own prediction/label pairs rather than in-
 1172 dependent hardware measurements. The per-device sample sizes
 1173 vary (3–18 configurations), limiting statistical power for devices
 1174 with few data points (e.g., P4 with only 3 configurations, A100-SXM
 1175 with 3 configurations). We mitigate this by reporting both mean
 1176 and worst-case APE. Our benchmark suite covers 28 scenarios, but
 1177 the distribution is not uniform: training scenarios (11) outnumber
 1178 inference scenarios (13), with MoE and multi-model scenarios (T4.4,
 1179 I4.1) represented by only one scenario each. A more balanced suite
 1180 might weight scenarios by practitioner frequency of use, but such
 1181 weighting data is not publicly available. Despite these limitations,
 1182 our suite provides the first standardized coverage metric for ML
 1183 performance tools, enabling future evaluations to quantitatively
 1184 compare tool ecosystems.

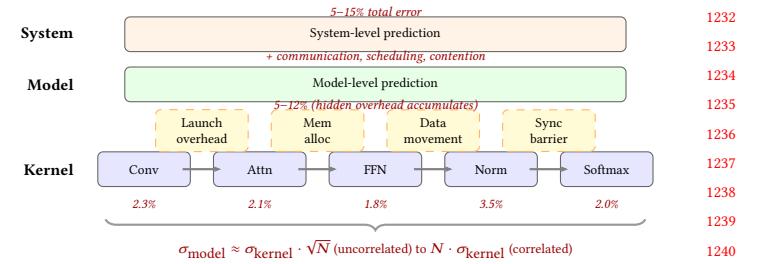
1185 **Construct validity.** Our approach prioritizes accuracy; tools
 1186 may provide value beyond this dimension (e.g., Timeloop’s energy
 1187 breakdown for design insight, ASTRA-sim’s what-if analysis for
 1188 topology exploration). The feature availability matrix partially ad-
 1189 dresses this, but our evaluation is designed to challenge accuracy
 1190 claims rather than comprehensively assess utility. Additionally,
 1191 our coverage criterion (supported/partial/unsupported) does not
 1192 capture the quality of partial support—ASTRA-sim’s partial cover-
 1193 age of MoE training (T4.4), for example, provides All-to-All com-
 1194 munication modeling but misses expert load balancing effects. A
 1195 finer-grained coverage metric—e.g., percentage of scenario-relevant
 1196 computations that a tool can model—would better capture partial
 1197 support quality but requires scenario-specific decomposition be-
 1198 yond our current scope.

1199 **Temporal validity.** Our evaluation reflects tool state as of Jan-
 1200 uary 2026. Tools under active development (ASTRA-sim, VIDUR,
 1201 NeuSight) may have addressed some identified limitations in sub-
 1202 sequent releases. However, our core findings about structural cov-
 1203 erage gaps and accuracy overstatement reflect fundamental design
 1204 choices rather than fixable bugs, and are likely to persist across ver-
 1205 sions. We encourage future evaluations to adopt our independent
 1206 verification methodology and benchmark suite to enable longitu-
 1207 dinal tracking of tool accuracy. The benchmark suite itself should
 1208 evolve as new LLM techniques emerge; we provide it as a living
 1209 document in the supplementary material.

1210 **Benchmark suite validity.** Our 28-scenario benchmark suite
 1211 was designed around the LLM workload landscape as of early 2026.
 1212 Emerging techniques not represented include retrieval-augmented
 1213 generation (RAG), which introduces variable-length retrieval la-
 1214 tency into the inference pipeline; multi-modal models combining
 1215 vision encoders with language models, which create heterogeneous



1216 **Figure 6: Unified five-layer pipeline. Layer 3 (dashed)**
 1217 **is the critical unmodeled gap.**



1218 **Figure 7: Error composition: kernel predictions (2–3%) accu-**
 1219 **mulate to 5–15% at system level.**

1220 compute patterns; and reinforcement learning from human feed-
 1221 back (RLHF), which requires modeling reward model inference
 1222 interleaved with policy updates. We designed the suite to be ex-
 1223 tensible: each scenario is specified by a tuple of (model architec-
 1224 ture, hardware configuration, parallelism strategy, target metric),
 1225 allowing new scenarios to be added as techniques mature without
 1226 restructuring the evaluation framework. Future versions should
 1227 expand to at least 40 scenarios to maintain coverage as the LLM
 1228 deployment landscape diversifies.

1229 8 Toward a Unified Simulation Pipeline

1230 No single tool spans kernel execution through serving SLAs. Figure
 1231 6 shows five layers where 5–9% kernel MAPE grows to 10–28%
 1232 at model level, driven by (i) interface heterogeneity, (ii) calibration
 1233 mismatch between steady-state models and transient-dominated
 1234 kernels, and (iii) feedback loops in serving schedulers.

1235 9 Open Challenges and Future Directions

- 1236 **(1) Composition gap:** Kernel errors of 2–3% yield 5–12% model-
 1237 level error (Figure 7) with no validated pipeline. **(2) Frontier work-
 1238 loads:** MoE, diffusion [40], and dynamic inference lack validated
 1239 tools; scaling laws [14, 22, 27, 37] predict loss but not latency (Figure
 1240 8). **(3) Hardware transfer:** Cross-family transfer (GPU→TPU→PIM [26,
 1241 31, 46, 58]) and congestion modeling [49, 84] remain unsolved.
 1242 **(4) Standardized evaluation:** No MLPerf [53, 67, 68] equivalent
 1243 exists for simulators; portable formats [74] and continuous valida-
 1244 tion are needed. **(5) Reproducibility:** nn-Meter failed from depen-
 1245 dency rot; containerization and CI testing are needed. **(6) Software**

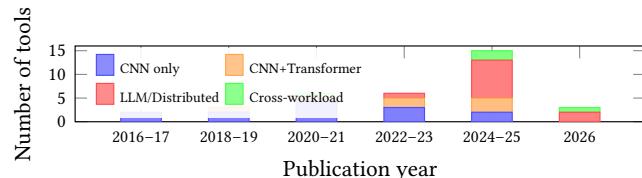


Figure 8: Workload coverage by publication period. MoE and diffusion models remain uncharacterized.

stack evolution: Rapidly evolving optimizations such as FlashAttention [16] invalidate performance models trained on prior kernel implementations.

10 Conclusion

We survey 22 ML performance tools and evaluate five against a 28-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/10.1177/10943420001400303> 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 (ASPOLOS)*. 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, Zheha 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] Diddharth 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 Hagelé, 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] Amerer Haj-Ali et al. 2025. Omnipwise: 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 (AS- PLOS)*. 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. Ne-upIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (AS- PLOS)*. 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*

- 1393 Computer Architecture (ISCA). 753–766. <https://doi.org/10.1109/ISCA59077.2024.00064> 1451
- 1394 [29] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu 1452
1395 Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng 1453
1396 Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline 1454
1397 Parallelism. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1455
1398 Vol. 32, 103–112. 1456
- 1399 [30] Rodrigo Huerta, Mojtaba Abaie Shoushtary, Jose-Lorenzo Cruz, and Antonio 1457
1400 Gonzalez. 2025. Dissecting and Modeling the Architecture of Modern GPU Cores. 1458
1401 In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 369–384. Reverse-engineers modern NVIDIA GPU cores, improves 1459
1402 Accel-Sim to 13.98% MAPE. UPC Barcelona. 1460
- 1403 [31] Bongjoon Hyun, Taejun Kim, Dongjae Lee, and Minsoo Rhu. 2024. Pathfinding 1461
1404 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 1462
1405 framework for UPMEM. KAIST. 1463
- 1406 [32] Ryota Imai, Kentaro Harada, Ryo Sato, and Toshio Nakaike. 2024. Roofline- 1464
1407 Driven Machine Learning for Large Language Model Performance Prediction. 1465
1408 *NeurIPS Workshop on Machine Learning for Systems* (2024). 1466
- 1409 [33] Anand Jayaraman, Wei-Lin Hu, Gauri Zhao, and Gennady Pekhimenko. 2023. 1467
1410 Sia: Heterogeneity-aware, Goodput-optimized ML-Cluster Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 642–657. 1468
1411 <https://doi.org/10.1145/3600006.3613175> Extends goodput optimization to het- 1469
1412 erogeneous GPU clusters for training workloads. 1470
- 1413 [34] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, 1471
1414 James Laudon, Cliff Young, and David Patterson. 2023. TPU v4: An Optically 1472
1415 Reconfigurable Supercomputer for Machine Learning with Hardware Support for 1473
1416 Embeddings. *Proceedings of the 50th Annual International Symposium on Computer 1474
1417 Architecture (ISCA)* (2023), 1–14. <https://doi.org/10.1145/3579371.3589350> 4096- 1475
1418 chip pods with 3D optical interconnect; up to 1.7x/2.1x faster than TPU v3. 1476
- 1419 [35] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, 1477
1420 Ramindei Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borber, et al. 2017. 1478
1421 In-Datcenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*. 1–12. 1479
1422 <https://doi.org/10.1145/3079856.3080246> First dedicated ML inference 1480
1423 accelerator; 15–30x over CPUs/GPUs on CNN inference. 1481
- 1424 [36] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios 1482
1425 Xydis, and Dimitrios Soudris. 2025. throtLL'eM: Predictive GPU Throttling for 1483
1426 Energy Efficient LLM Inference Serving. In *Proceedings of the IEEE International 1484
1427 Symposium on High Performance Computer Architecture (HPCA)*. 1–14. Achieves 1485
1428 up to 43.8% lower energy consumption for LLM inference. 1486
- 1429 [37] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, 1487
1430 Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. 1488
1431 Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361* 1489
1432 (2020). Original neural scaling laws: power-law relationships between model 1490
1433 size, dataset size, compute, and loss. 1491
- 1434 [38] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. 1492
1435 Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In 1493
1436 *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*. 1494
1437 473–486. <https://doi.org/10.1109/ISCA45697.2020.00047> 1495
- 1438 [39] Jungho Kim et al. 2025. PyTorchSim: A Comprehensive, Fast, and Accurate 1496
1439 NPU Simulation Framework. In *Proceedings of the 58th IEEE/ACM International 1497
1440 Symposium on Microarchitecture (MICRO)*. 1–14. <https://doi.org/10.1145/3725843.3756045> 1498
1441 PyTorch 2-integrated NPU simulator with custom RISC-V ISA and Tile-Level 1499
1442 Simulation. 1500
- 1443 [40] Jin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. 2026. The Cost of 1501
1444 Dynamic Reasoning: Demystifying AI Agents and Test-Time Scaling from an AI 1502
1445 Infrastructure Perspective. In *Proceedings of the IEEE International Symposium 1503
1446 on High Performance Computer Architecture (HPCA)*. 1–14. HPCA 2026 (Jan 31– 1504
1447 Feb 4, 2026, Las Vegas). First comprehensive system-level analysis of AI agents; 1505
1448 quantifies resource usage, latency, and datacenter power consumption. 1506
- 1449 [41] Yoongi Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and 1507
1450 Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45– 1508
- 1451 49. <https://doi.org/10.1109/LCA.2015.2414456> Fast extensible DRAM simulator 1509
1452 supporting DDRx, LPDDRx, GDDRx, WIOx, HBMx standards. 1510
- 1453 [42] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Norman P. 1511
1454 Jouppi, Jignesh Parmar, Hyoukjun Kim, James Laudon, and Chandrakant 1512
1455 Narayanaswami. 2023. ArchGym: An Open-Source Gymnasium for Machine 1513
1456 Learning Assisted Architecture Design. In *Proceedings of the 50th International 1514
1457 Symposium on Computer Architecture (ISCA)*. 1–16. <https://doi.org/10.1145/3579371.3589049> 1515
- 1458 [43] Hyoukjun Kwon, Prasanth Chatarasi, Michael Sarber, Michael Pellauer, Angshuman 1516
1459 Parashar, and Tushar Krishna. 2019. MAESTRO: A Data-Centric Approach 1517
1460 to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. In 1518
1461 *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14. <https://doi.org/10.1145/3352460.3358292> 1519
- 1462 [44] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, 1520
1463 Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient 1521
1464 Memory Management for Large Language Model Serving with PagedAttention. 1522
1465 In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 1523
1466 611–626. <https://doi.org/10.1145/3600006.3613165> 1524
- 1467 [45] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, 1525
1468 Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr 1526
1469 Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific 1527
1470 Computation. In *Proceedings of the IEEE/ACM International Symposium on Code 1528
1471 Generation and Optimization (CGO)*. 2–14. <https://doi.org/10.1109/CGO51591.2021.9370308> 1529
1472 Multi-level IR infrastructure enabling cost model composition across abstraction 1530
1473 levels. 1531
- 1474 [46] Hyojung Lee, Daehyeon Baek, Jimyoung Son, Jieun Choi, Kihyo Moon, and 1532
1475 Minsung Jang. 2025. PAISE: PIM-Accelerated Inference Scheduling Engine for 1533
1476 Transformer-based LLM. In *Proceedings of the IEEE International Symposium on 1534
1477 High Performance Computer Architecture (HPCA)*. 1–14. PIM-based LLM 1535
1478 inference scheduling, 48.3% speedup, 11.5% power reduction. Samsung. 1536
- 1479 [47] Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. 2021. HELP: 1537
1480 Hardware-Adaptive Efficient Latency Prediction for NAS via Meta-Learning. 1538
1481 In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34. 27016– 1539
1482 27028. 1540
- 1483 [48] Seunghyun Lee, Amar Phanishayee, and Divya Mahajan. 2025. NeuSight: GPU 1541
1484 Performance Forecasting via Tile-Based Execution Analysis. In *Proceedings of the 1542
1485 the 30th ACM International Conference on Architectural Support for Programming 1543
1486 Languages and Operating Systems (ASPLOS)*. 1–15. 1544
- 1487 [49] Jianbo Li et al. 2025. TrioSim: A Lightweight Simulator for Large-Scale DNN 1545
1488 Workloads on Multi-GPU Systems. In *Proceedings of the 52nd Annual International 1546
1489 Symposium on Computer Architecture (ISCA)*. 1–13. Multi-GPU DNN simulation 1547
1490 with lightweight approach for distributed training analysis. 1548
- 1491 [50] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. 1549
1492 DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer 1550
1493 Architecture Letters* 19, 2 (2020), 106–109. <https://doi.org/10.1109/LCA.2020.2973991> 1551
1494 Modernized DRAM simulator with thermal modeling and HMC support. 1552
- 1495 [51] Wenxuan Liang et al. 2025. Lumos: Efficient Performance Modeling and Estimation 1553
1496 for Large-scale LLM Training. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–16. Trace-driven performance modeling achieving 3.3% error 1554
1497 on H100 GPUs for LLM training. 1555
- 1498 [52] Haocong Luo, Yahya Can Tugrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray 1556
1499 Yaglkci, and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible 1557
1500 DRAM Simulator. *IEEE Computer Architecture Letters* 22, 2 (2023), 129–132. 1558
1501 <https://doi.org/10.1109/LCA.2023.3333759> Modular DRAM simulator with DDR5, 1559
1502 LPDDR5, HBM3, GDDR6 support and RowHammer mitigation modeling. 1560
- 1503 [53] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Mickevicius, 1561
1504 David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2020. 1562
1505 MLPPerf Training Benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*. 336–349. Standard ML training benchmark suite covering 1563
1506 image classification, object detection, NLP, recommendation, reinforcement 1564
1507 learning. 1565
- 1508 [54] Azaz-Ur-Rehman Nasir, Samroz Ahmad Shoib, Muhammad Abdullah Hanif, and 1566
1509 Muhammad Shafique. 2025. ESM: A Framework for Building Effective Surrogate 1567
1510 Models for Hardware-Aware Neural Architecture Search. In *Proceedings of the 1568
1511 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–6. 97.6% accuracy 1569
1512 surrogate model framework for HW-aware NAS. 1570
- 1513 [55] Amir Nasr-Esfahany et al. 2025. Concorde: Fast and Accurate CPU Performance 1571
1514 Modeling with Compositional Analytical-ML Fusion. In *Proceedings of the 52nd 1572
1515 Annual International Symposium on Computer Architecture (ISCA)*. 1–15. Hybrid 1573
1516 analytical-ML approach achieving 2% CPI error at 5 orders of magnitude faster 1574
1517 than gem5. 1575
- 1518 [56] NVIDIA Corporation. 2019. Nsight Compute: Interactive Kernel Profiler. <https://developer.nvidia.com/nsight-compute>. Industry-standard GPU kernel profiling 1576
1519 tool with roofline analysis. 1577
- 1520 [57] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, 1578
1521 Victor A. Ying, Anurag Muber, Rangharajan Venkatesan, Brueckl Khalilany, 1579
1522 Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach 1580
1523 to DNN Accelerator Evaluation. In *Proceedings of the IEEE International Symposium on 1581
1524 Performance Analysis of Systems and Software (ISPASS)*. 304–315. 1582
1525 <https://doi.org/10.1109/ISPASS.2019.00042> 1583
- 1526 [58] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk 1584
1527 Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of 1585
1528 PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of the 1586
1529 29th ACM International Conference on Architectural Support for Programming Languages and 1587
1530 Operating Systems (ASPLOS)*. 1–16. PIM-based accelerator for batched transformer 1588
1531 attention. Seoul National University/UIUC.. 1589
- 1532 [59] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory 1590
1533 Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. 1591
1534 PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural 1592
1535 Information Processing Systems (NeurIPS)*, Vol. 32. 8024–8035. 1593

- [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 Jayaraman, 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, Frédéric 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 Breeshevov, 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] 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
- [70] 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.
- [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] Zhumo 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 Phothilmatha, 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 Tschanz, 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/MICRO56248.2022.000078>
- [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 Lyra 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, Amer 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

1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624

1625	for Goodput-optimized Large Language Model Serving. In <i>Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)</i> .	1–18.	1683
1626			1684
1627			1685
1628			1686
1629			1687
1630			1688
1631			1689
1632			1690
1633			1691
1634			1692
1635			1693
1636			1694
1637			1695
1638			1696
1639			1697
1640			1698
1641			1699
1642			1700
1643			1701
1644			1702
1645			1703
1646			1704
1647			1705
1648			1706
1649			1707
1650			1708
1651			1709
1652			1710
1653			1711
1654			1712
1655			1713
1656			1714
1657			1715
1658			1716
1659			1717
1660			1718
1661			1719
1662			1720
1663			1721
1664			1722
1665			1723
1666			1724
1667			1725
1668			1726
1669			1727
1670			1728
1671			1729
1672			1730
1673			1731
1674			1732
1675			1733
1676			1734
1677			1735
1678			1736
1679			1737
1680			1738
1681			1739
1682			1740