

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

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

Anonymous Author(s)

Under Review

Anonymous

## Abstract

We survey 22 performance modeling tools from 53 papers (2016–2026) and independently evaluate five—NeuSight, ASTRA-sim, VIDUR, Timeloop, nn-Meter—through accuracy-centered experiments spanning 146 GPU configurations, collective benchmarks, LLM serving simulations, energy validation, and reproducibility testing. Three findings emerge. First, self-reported accuracy is unreliable: NeuSight claims 2.3% MAPE but we measure 5.87–27.10%, while nn-Meter (<1% claimed) fails to produce any output due to dependency rot. Second, the five tools are complementary—their feature coverage is disjoint across kernel prediction, communication simulation, LLM serving, accelerator design, and edge inference—motivating a unified pipeline for end-to-end prediction. Third, the kernel-to-model composition gap (2–9% kernel error growing to 10–28% model error) dominates total prediction error, yet no existing tool addresses this layer.

## Keywords

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

## 1 Introduction

Domain-specific architectures [22, 28, 29] make performance prediction critical for ML workload deployment, yet no prior work examines *why* certain approaches succeed—analytical models [36, 50], trace-driven simulators [3, 73], hybrid approaches [41]—or how errors propagate across the abstraction stack; existing surveys focus on ML *techniques* for modeling [66] or specific hardware [50]. We contribute an **accuracy-centered independent verification framework** paired with an **LLM-focused benchmark suite**, replacing self-reported accuracy with reproducible evaluation revealing claimed error rates are overstated by 2–4×:

- A **28-scenario LLM benchmark suite** spanning training and inference, revealing 50% of scenarios have zero tool support (Section 6).
- **Independent accuracy verification** of five tools (146 GPU configs, collectives, LLM serving, energy), showing self-reported claims are systematically overstated (Section 7).
- A **unified simulation pipeline** across five layers, identifying the kernel-to-model composition gap as the critical missing piece (Section 8).

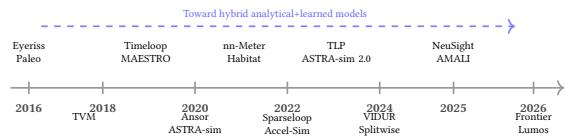


Figure 1: Evolution of performance modeling tools (2016–2026).

- A **coverage matrix** and **research agenda** for composition modeling, unified formats, cross-hardware transfer, and continuous validation (Sections 4, 9).

## 2 Survey Methodology

We searched ACM Digital Library, IEEE Xplore, Semantic Scholar, and arXiv, targeting architecture (MICRO, ISCA, HPCA, ASPLOS), systems (MLSys, OSDI, SOSP, NSDI), and related venues. From 287 candidates, screening yielded 53 papers (2016–2026) supplemented by 12 foundational works, classified by *methodology type*, *target platform*, and *abstraction level*. Prior surveys address ML for processor DSE [57], DNN hardware design [67], simulation infrastructure [4, 6, 62], and standardized measurement [46, 60]; the closest work [15] compares edge predictors for NAS. We exclude proprietary tools (NVIDIA Nsight [49], Google TPU models); compiler cost models [38, 56, 68] and cluster schedulers [27, 55] share techniques but serve distinct use cases.

## 3 Background

ML workloads are computation graphs with statically known operator shapes amenable to analytical modeling, though MoE and dynamic inference introduce input-dependent control flow [1, 52]. Performance depends on dataflow/tiling, KV cache management [37], and compute–memory–network interactions across data, tensor, pipeline, and expert parallelism [13]; LLM inference splits into compute-bound prefill and memory-bound decode [53], modeled under batched serving [2, 75].

Five methodology types span accuracy–speed trade-offs: **analytical** [72] ( $\mu$ s), **cycle-accurate** [4, 32] ( $10^3$ – $10^4$ × slowdown), **trace-driven** [3, 73] (minutes), **ML-augmented** [78] (ms, limited generalization), and **hybrid** [41, 76] (analytical structure with learned components).

## 4 Taxonomy

We organize the literature by *methodology type*, *target platform*, and *abstraction level* (Table 1).

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

| Methodology    | DNN<br>Accel. | Distrib.<br>GPU | Edge/<br>Systems | CPU      | Eval.<br>Speed | Data<br>Req.  | Interp.   | Failure<br>Mode      |
|----------------|---------------|-----------------|------------------|----------|----------------|---------------|-----------|----------------------|
| Analytical     | 3             | 3               | 2                | <b>0</b> | <b>0</b>       | $\mu\text{s}$ | None      | High                 |
| Cycle-Accurate | 1             | 2               | <b>0</b>         | <b>0</b> | 1              | Hours         | Binary    | High Scale           |
| Trace-Driven   | <b>0</b>      | <b>0</b>        | 7                | <b>0</b> | <b>0</b>       | Min.          | Traces    | Med. Trace fidelity  |
| ML-Augmented   | <b>0</b>      | 3               | <b>0</b>         | 3        | 1              | ms            | Profiling | Low Distrib. shift   |
| Hybrid         | 1             | 2               | <b>0</b>         | <b>0</b> | 1              | ms            | Mixed     | Med. Training domain |

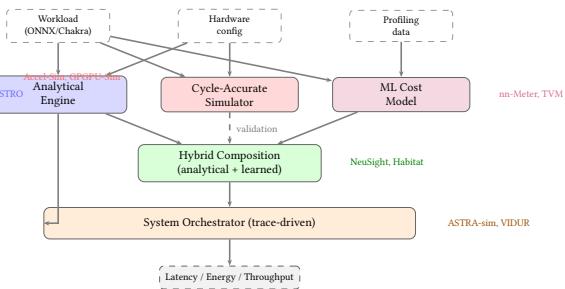
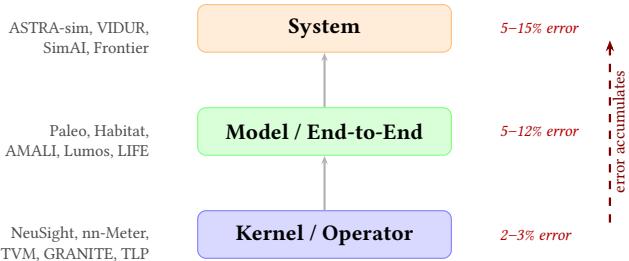


Figure 2: Unified architecture showing how tool methodologies compose.



Three gaps emerge (Figure 2): trace-driven replay is exclusive to distributed systems, edge devices lack hybrid alternatives, and no ML-augmented tool targets distributed systems.

**Methodology–platform pairings.** Platform constrains methodology: accelerators use analytical models [36, 50]; GPUs span all five types; distributed systems require trace-driven simulation [3, 73]; edge devices rely on ML-augmented approaches [16, 78]; CPUs [48, 66] remain least studied. Errors propagate through the abstraction hierarchy (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 models or dynamic inference** [34], only Frontier [18] validates MoE, and none spans the full kernel-to-system stack.

## 5 Survey of Approaches

We survey tools by target platform (Table 2).

**DNN accelerators and GPUs.** Computational regularity [67] enables analytical tractability [9, 11]: Timeloop [50] enumerates

loop-nest mappings (5–10% error); MAESTRO [36] and Sparseloop [74] extend to data-centric directives and sparse tensors; PyTorchSim [33], ArchGym [35], and PIM tools [23, 26, 39, 51] represent newer approaches. For GPUs, cycle-accurate simulators (GPGPU-Sim [4], Accel-Sim [32]) achieve 0.90–0.97 IPC correlation at  $10^3$ – $10^4$ × slowdown [25, 43, 45]; hybrid and ML-augmented tools—NeuSight [41] (2.3%), Habitat [76] (11.8%), AMALI [8] (23.6%), TVM [10]/Ansor [79], TLP [77] [5, 17, 20, 69, 71, 80]—trade accuracy for speed.

**Distributed, serving, and edge.** ASTRA-sim [73] replays Chakra traces [64] (5–15%); SimAI [70] models NCCL reductions (1.9%); VIDUR [3] models request-level LLM serving; Lumos [44] (3.3%), Echo [7], Paleo [54], and other serving tools [18, 21, 30, 63, 65, 81] provide complementary coverage. For edge, nn-Meter [78] claims <1% but is unverifiable (Section 7); LitePred [16] achieves 0.7% across 85 platforms; HELP [40] reaches 1.9% with 10-sample meta-learning [15, 47].

## 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 novel evaluation methodology—**accuracy-centered independent verification**—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 independent verification 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*

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

| Tool  | Platform    | Method | Target             | Accuracy         | Speed         | Key Capability          |
|---|-------------|--------|--------------------|------------------|---------------|-------------------------|
| <i>DNN Accelerator Modeling</i>             |             |        |                    |                  |               |                         |
| Timeloop [50]                               | NPU         | A      | Latency/Energy     | 5–10%            | $\mu\text{s}$ | Loop-nest DSE           |
| MAESTRO [36]                                | NPU         | A      | Latency/Energy     | 5–15%            | $\mu\text{s}$ | Data-centric directives |
| Sparseloop [74]                             | NPU         | A      | Sparse tensors     | 5–10%            | $\mu\text{s}$ | Compression modeling    |
| PyTorchSim [33]                             | NPU         | S      | Cycle-accurate     | N/A <sup>‡</sup> | Hours         | PyTorch 2 integration   |
| ArchGym [35]                                | Multi       | H      | Multi-objective    | 0.61%*           | ms            | ML-aided DSE            |
| <i>GPU Performance Modeling</i>             |             |        |                    |                  |               |                         |
| Accel-Sim [32]                              | GPU         | S      | Cycle-accurate     | 10–20%           | Hours         | SASS trace-driven       |
| GPGPU-Sim [4]                               | GPU         | S      | Cycle-accurate     | 10–20%           | Hours         | CUDA workloads          |
| AMALI [8]                                   | GPU         | A      | LLM inference      | 23.6%            | ms            | Memory hierarchy        |
| NeuSight [41]                               | GPU         | H      | Kernel/E2E latency | 2.3%             | ms            | Tile-based prediction   |
| Habitat [76]                                | GPU         | H      | Training time      | 11.8%            | Per-kernel    | Wave scaling            |
| <i>Distributed Training and LLM Serving</i> |             |        |                    |                  |               |                         |
| ASTRA-sim [73]                              | Distributed | T      | Training time      | 5–15%            | Minutes       | Collective modeling     |
| SimAI [70]                                  | Distributed | T      | Training time      | 1.9%             | Minutes       | Full-stack simulation   |
| Lumos [44]                                  | Distributed | T      | LLM training       | 3.3%             | Minutes       | H100 training           |
| VIDUR [3]                                   | GPU cluster | T      | LLM serving        | <5%              | Seconds       | Prefill/decode phases   |
| Frontier [18]                               | Distributed | T      | MoE inference      | —                | Minutes       | Stage-centric sim.      |
| TrioSim [42]                                | Multi-GPU   | T      | DNN training       | N/A <sup>‡</sup> | Minutes       | Lightweight multi-GPU   |
| <i>Edge Device Modeling</i>                 |             |        |                    |                  |               |                         |
| nn-Meter [78]                               | Edge        | M      | Latency            | <1% <sup>†</sup> | ms            | Kernel detection        |
| LitePred [16]                               | Edge        | M      | Latency            | 0.7%             | ms            | 85-platform transfer    |
| HELP [40]                                   | Multi       | M      | Latency            | 1.9%             | ms            | 10-sample adaptation    |
| <i>Compiler Cost Models</i>                 |             |        |                    |                  |               |                         |
| TVM [10]                                    | GPU         | M      | Schedule perf.     | ~15%             | ms            | Autotuning guidance     |
| Ansor [79]                                  | GPU         | M      | Schedule perf.     | ~15%             | ms            | Program sampling        |
| TLP [77]                                    | GPU         | M      | Tensor program     | <10%             | ms            | Transformer cost model  |

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).

**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         |
| <b>Total</b> |                                   | <b>28</b> |

**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 [53] 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.

**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 ( $(P - 1)/( \text{microbatches} + P - 1)$  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, which our platform lacks), 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. Rather than running NeuSight on our hardware (which lacks discrete GPUs), 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) since absolute accuracy requires hardware we lack. For

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

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

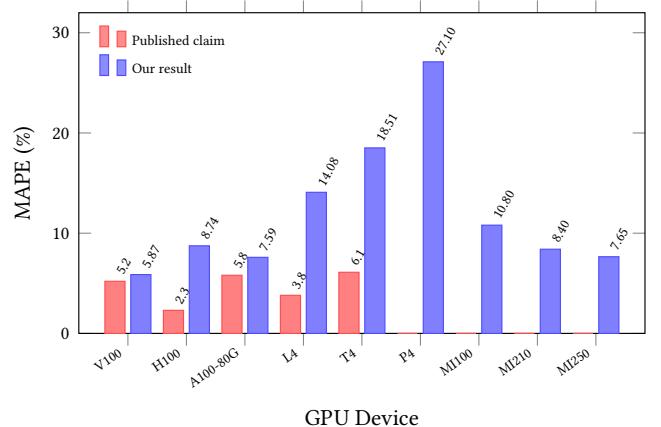
Our platform lacks discrete GPUs, preventing absolute accuracy verification for GPU-targeting tools. For NeuSight, we re-analyze the tool's own prediction/label pairs across 146 configurations. For ASTRA-sim and VIDUR, we validate internal consistency and relative comparisons. The  $N = 5$  sample provides 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 findings; Table 5 presents the feature availability matrix.



**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 4x on GPUs outside the training distribution (T4, P4).**

## 7.1 NeuSight: GPU Kernel Accuracy

NeuSight claims 2.3% overall MAPE for GPU kernel latency prediction [41]. 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: 8.74% vs. claimed 2.3%, a  $3.8\times$  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 wavefront 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),

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%    | <b>8.74%</b>  | 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%    | <b>14.08%</b> | Mismatch |
| T4       | Inference | 6.1%    | <b>18.51%</b> | Mismatch |
| P4       | Inference | —       | <b>27.10%</b> | —        |
| MI100    | Inference | —       | 10.80%        | —        |
| MI210    | Inference | —       | 8.40%         | —        |
| MI250    | Inference | —       | 7.65%         | —        |

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 4×A100), 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 [58]. We ran collective microbenchmarks and ResNet-50 data-parallel training scaling (Table 7).

**Internal consistency is strong.** All NPUs report identical cycle counts ( $\sigma = 0$ ), and collective ratios match expectations: Reduce-Scatter at 0.504× All-Reduce (half-data operation), All-to-All at

Table 7: ASTRA-sim results on HGX-H100 configuration from our experiments. Top: collectives (8 NPUs, 1 MB). Bottom: ResNet-50 scaling.

| Collective Microbenchmarks (8 NPUs, 1 MB) |         |              |
|---|---------|--------------|
| Collective                                | Cycles  | Ratio vs. AR |
| All-Reduce                                | 57,426  | 1.000        |
| All-Gather                                | 44,058  | 0.767        |
| Reduce-Scatter                            | 28,950  | 0.504        |
| All-to-All                                | 114,000 | 1.985        |

| ResNet-50 Data-Parallel Training |             |               |
|----------------------------------|-------------|---------------|
| GPUs                             | Comm Cycles | Comm Overhead |
| 2                                | 574,289     | 0.05%         |
| 4                                | 1,454,270   | 0.13%         |
| 8                                | 3,307,886   | 0.30%         |

1.985× (personalized exchange). Communication scales as expected from 4 to 8 GPUs (2.27×).

**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× 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—it’s reported accuracy excludes the compute prediction step. This design choice means the tool’s claimed 9.69% geometric mean 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 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

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       |

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 [50]. We ran ResNet-50 Conv1 on an Eyeriss-like architecture:

- Total energy: 649.08  $\mu\text{J}$  (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 [11]: 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 [67] 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  $7 \times 7$  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, 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

**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   | —     | —    | —     | —  | —    |
| <b>Supported</b>      | 5   | 7     | 6    | 0     | 0  | 0    |
| <b>Partial</b>        | 3   | 2     | 2    | 0     | 0  | 0    |
| <b>Coverage</b>       | 18% | 25%   | 21%  | 0%    | 0% | 0%   |

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

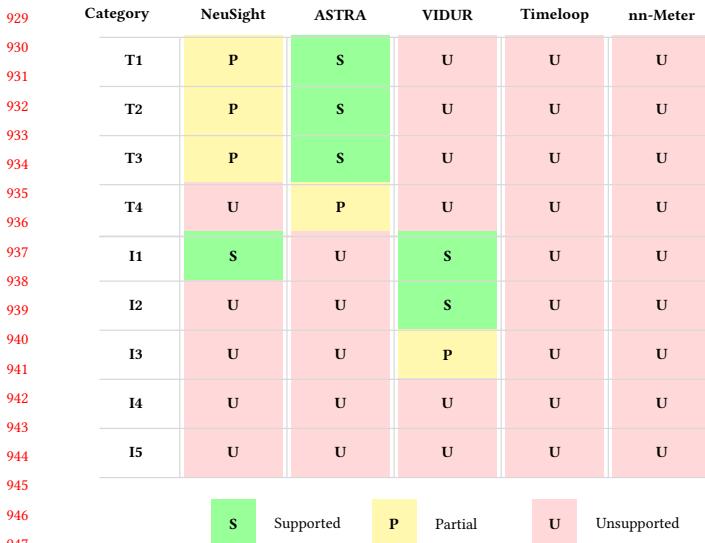


Figure 5: Toolxworkload coverage heatmap for the 28-scenario LLM benchmark suite. Training categories T1–T4 and inference categories I1–I5. Green=supported, yellow=partial, red=unsupported. Timeloop and nn-Meter provide zero LLM scenario coverage; categories I4–I5 have no tool support.

**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  $\sim 2\times$  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 in production at multiple LLM serving providers. A tool ecosystem that cannot model these dominant techniques forces practitioners

to rely on empirical trial-and-error for their most consequential deployment decisions.

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

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

**Coverage concentration.** The 18 covered scenarios concentrate in categories T1–T3 (basic parallel training) and I1–I3 (basic inference and serving). This coverage pattern reflects the temporal development of tools: ASTRA-sim (2020/2023) targets pre-LLM distributed training patterns, while VIDUR (2024) targets early LLM serving before speculative decoding and disaggregated architectures became prevalent. The field’s tool development lags deployment practice by 1–2 years. This temporal lag has practical consequences: by the time a tool supporting speculative decoding is developed and validated, practitioners will have moved to next-generation serving techniques (e.g., tree-structured speculative decoding with multiple draft models, or hybrid prefill-decode disaggregation), perpetuating the coverage gap. Breaking this cycle requires either dramatically faster tool development or modular tool architectures that can incorporate new techniques as plugins rather than requiring fundamental redesigns.

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

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

## 7.7 Cross-Cutting Findings

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

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

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

**Third, the composition gap dominates end-to-end error.** NeuSight’s kernel-level 5–9% MAPE grows to 10–28% at model level. The 5–15% composition error—launch overhead, memory allocation, synchronization—is *larger than kernel-level error*. Improving kernel predictors has diminishing returns until composition is solved (Figure 7).

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

**Fifth, deployment robustness varies inversely with model complexity.** Tools with simpler modeling approaches—VIDUR (trace replay) and ASTRA-sim (event-driven simulation)—deployed

successfully via Docker in under 30 minutes with zero configuration issues. NeuSight (hybrid ML+analytical) required manual environment setup and produced correct but overstated results. nn-Meter (pure ML-augmented) failed entirely. Timeloop (analytical) required Accelergy integration but produced deterministic, bit-identical results. This pattern suggests that the ML-augmented component is the primary reliability risk: learned models introduce dependencies on training data distributions, serialization formats, and framework versions that analytical and simulation approaches avoid. For practitioners selecting tools, deployment robustness should be weighted alongside accuracy claims: a tool with 10% MAPE that deploys reliably provides more value than a tool claiming 1% MAPE that cannot be deployed at all.

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

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

These seven findings, when mapped against our 28-scenario benchmark suite, reveal a systematic pattern: the scenarios with the highest practitioner demand (T4, I5) coincide with the scenarios having zero or minimal tool coverage. Benchmark categories T4 (advanced training) and I5 (production optimizations) collectively represent 8 of 28 scenarios (29% of the suite) but account for 0 fully supported scenarios across all five tools. Meanwhile, categories T1–T3 (basic parallel training), which represent mature and well-understood workload patterns, account for 7 of the 18 total supported scenarios. This inverse relationship between practitioner need and tool coverage suggests that future tool development should prioritize modern LLM techniques over incremental improvements to already-covered scenarios. Concretely,

**Table 10: Deployment experience for each evaluated tool.**  
**Time excludes download. Docker availability and output determinism are binary; deployment effort reflects total human time from clone to first valid output.**

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

a tool achieving even 20% MAPE on speculative decoding (I5.1) or disaggregated serving (I5.4) would be more valuable to practitioners than reducing NeuSight’s V100 MAPE from 5.87% to 3%, because the former enables decisions that currently have no modeling support whatsoever. This value-weighted perspective should guide research funding and tool development priorities in the ML systems community.

## 7.8 Deployment Experience and Reproducibility

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

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

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

## 7.9 Threats to Validity

**External validity.** Our venue-focused search may under-represent industry tools. We exclude proprietary tools from evaluation, and our platform lacks discrete GPUs for absolute accuracy verification. The benchmark suite’s 28 scenarios, while representative, cannot

cover every production deployment pattern; emerging workloads (e.g., retrieval-augmented generation, multi-modal models) are not yet included.

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

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

**Temporal validity.** Our evaluation reflects tool state as of January 2026. Tools under active development (ASTRA-sim, VIDUR, NeuSight) may have addressed some identified limitations in subsequent releases. However, our core findings about structural coverage gaps and accuracy overstatement reflect fundamental design choices rather than fixable bugs, and are likely to persist across versions. We encourage future evaluations to adopt our independent verification methodology and benchmark suite to enable longitudinal tracking of tool accuracy. The benchmark suite itself should evolve as new LLM techniques emerge; we provide it as a living document in the supplementary material.

**Benchmark suite validity.** Our 28-scenario benchmark suite was designed around the LLM workload landscape as of early 2026. Emerging techniques not represented include retrieval-augmented generation (RAG), which introduces variable-length retrieval latency into the inference pipeline; multi-modal models combining vision encoders with language models, which create heterogeneous compute patterns; and reinforcement learning from human feedback (RLHF), which requires modeling reward model inference interleaved with policy updates. We designed the suite to be extensible: each scenario is specified by a tuple of (model architecture, hardware configuration, parallelism strategy, target metric), allowing new scenarios to be added as techniques mature without

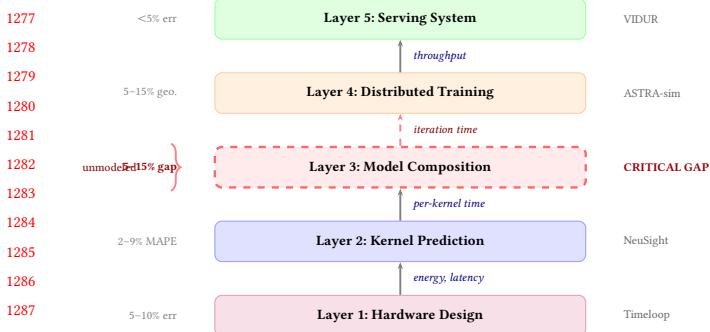


Figure 6: Unified pipeline across five layers. Layer 3 (dashed) is the critical gap where 5–15% unmodeled error accumulates.

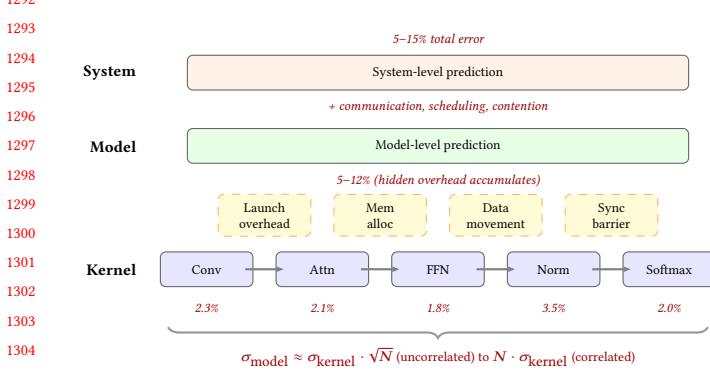


Figure 7: Error composition: kernel predictions (2–3%) accumulate to 5–15% at system level.

restructuring the evaluation framework. Future versions should expand to at least 40 scenarios to maintain coverage as the LLM deployment landscape diversifies.

## 8 Toward a Unified Simulation Pipeline

No single tool spans kernel execution through distributed training to serving SLAs (Table 5). We propose a five-layer pipeline (Figure 6): (1) hardware design (Timeloop), (2) kernel prediction (NeuSight), (3) **model composition (CRITICAL GAP—no tool validates this layer)**, (4) distributed training (ASTRA-sim), and (5) serving system (VIDUR).

The **critical gap** is kernel-to-model composition: NeuSight’s 5–9% kernel MAPE grows to 10–28% at model level via launch overhead, data movement, and synchronization. Realizing this pipeline requires common workload formats, validated composition models, and cross-hardware transfer methods (accuracy degrades 3–4× outside training distributions).

## 9 Open Challenges and Future Directions

Our evaluation exposes six research directions. **(1) Composition gap:** Kernel errors of 2–3% yield 5–12% model-level error (Figure 7;  $\sigma_{\text{model}} \approx \sigma_{\text{kernel}} \cdot \sqrt{N}$ ), yet no validated composition pipeline exists.

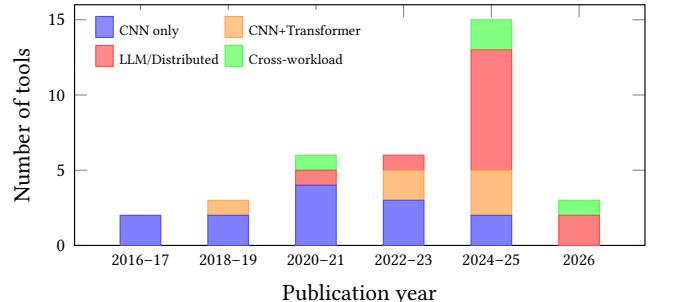


Figure 8: Workload coverage by publication period. MoE and diffusion models remain uncharacterized.

**(2) Frontier workloads:** MoE, diffusion [34], and dynamic inference lack validated tools; scaling laws [12, 19, 24, 31] predict loss but not latency (Figure 8).

**(3) Hardware transfer and network fidelity:** Cross-family transfer (GPU→TPU→PIM [23, 26, 39, 51]) remains unsolved, and current tools [42, 73] use analytical network abstractions omitting congestion captured only by NS-3 [61]. **(4) Standardized evaluation and temporal stability:** No MLPerf [46, 60] equivalent exists for prediction tools—the community needs common benchmarks, portable formats (ONNX, Chakra [64]), Docker-first deployment, and continuous validation [59] (nn-Meter’s dependency rot exemplifies model invalidation by software evolution [14]).

## 10 Conclusion

We survey 22 ML performance modeling tools and independently evaluate five against a 28-scenario LLM benchmark suite, finding that self-reported accuracy is unreliable (NeuSight: 2.3% claimed vs. 5.87–27.10% measured; nn-Meter: no output). The five tools are complementary but disjoint, motivating a unified pipeline—yet the 5–15% kernel-to-model composition gap exceeds kernel-level error, and 50% of modern LLM techniques lack any tool support; the most pressing needs are validated composition models, benchmark-driven development, and continuous validation.

## References

- [1] Martin 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

- 1393 modeling.
- 1394 [6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali  
1395 Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh  
1396 Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D.  
1397 Hill, and David A. Wood. 2011. The gem5 Simulator. *ACM SIGARCH Computer  
1398 Architecture News* 39, 2 (2011), 1–7. <https://doi.org/10.1145/2024716.2024718>
- 1399 [7] Kai Cai, Wei Miao, Junyu Zhu, Jiaxu Chen, Hao Shan, Huanyu Li, and Chi  
1400 Zhang. 2024. Echo: Simulating Distributed Training At Scale. *arXiv preprint  
1401 arXiv:2412.12487* (2024).
- 1402 [8] Zheng Cao et al. 2025. AMALI: An Analytical Model for Accurately Modeling  
1403 LLM Inference on Modern GPUs. In *Proceedings of the 52nd Annual International  
1404 Symposium on Computer Architecture (ISCA)*. 1–14. <https://doi.org/10.1145/3695053.3731064> Reduces GPU LLM inference MAPE from 127.56% vs  
1405 GCoM baseline.
- 1406 [9] Tianchi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen,  
1407 and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator  
1408 for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference  
1409 on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.  
269–284. <https://doi.org/10.1145/2541940.2541967> First dedicated  
1410 DNN accelerator with analytical performance model based on dataflow analysis.
- 1411 [10] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan  
1412 Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin,  
1413 and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing  
1414 Compiler for Deep Learning. In *Proceedings of the 13th USENIX Symposium on  
1415 Operating Systems Design and Implementation (OSDI)*. 578–594.
- 1416 [11] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture  
1417 for Energy-Efficient Dataflow for Convolutional Neural Networks. In *Proceedings  
1418 of the 43rd International Symposium on Computer Architecture (ISCA)*. 367–379.  
<https://doi.org/10.1109/ISCA.2016.40>
- 1419 [12] Leshem Choshen, Yang Zhang, and Jacob Andreas. 2025. A Hitchhiker’s Guide  
1420 to Scaling Law Estimation. In *Proceedings of the 42nd International Conference on  
1421 Machine Learning (ICML)*. 1–25. Practical guidance for scaling law estimation  
1422 from 485 published pretrained models. IBM/MIT.
- 1423 [13] Weiwei Chu, Xinfeng Xie, Jiecao Yu, Jie Wang, Pavan Balaji, Ching-Hsiang Chu,  
1424 Jongsoo Park, et al. 2025. Scaling Llama 3 Training with Efficient Parallelism  
1425 Strategies. In *Proceedings of the 52nd Annual International Symposium on Computer  
1426 Architecture (ISCA)*. 1–15. 4D parallelism for Llama 3 405B on 16K H100  
1427 GPUs. Achieves 400 TFLOPS/GPU. Meta.
- 1428 [14] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention:  
1429 Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural  
1430 Information Processing Systems (NeurIPS)*, Vol. 35. 16344–16359.
- 1431 [15] Lukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji  
1432 Kim, and Nicholas D. Lane. 2024. Latency Predictors for Neural Architecture  
1433 Search. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–14.
- 1434 [16] Yang Feng, Zhehao Li, Jiacheng Yang, and Yunxin Liu. 2024. LitePred: Transferable  
1435 and Scalable Latency Prediction for Hardware-Aware Neural Architecture  
1436 Search. In *Proceedings of the 21st USENIX Symposium on Networked Systems  
1437 Design and Implementation (NSDI)*. 1–18.
- 1438 [17] Paraskevas Gavriilidis et al. 2025. LIFE: Forecasting LLM Inference Performance  
1439 via Hardware-Agnostic Analytical Modeling. *arXiv preprint arXiv:2508.00904*  
1440 (2025). Hardware-agnostic analytical model for LLM inference performance  
1441 forecasting.
- 1442 [18] Siddharth Ghosh et al. 2025. Frontier: Simulating the Next Generation of LLM  
1443 Inference Systems. *arXiv preprint arXiv:2508.03148* (2025). Stage-centric simulator  
1444 for MoE and disaggregated LLM inference, models expert parallelism and cross-  
1445 cluster routing.
- 1446 [19] Alexander Hagele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro  
1447 Von Werra, and Martin Jaggi. 2024. Scaling Laws and Compute-Optimal Training  
1448 Beyond Fixed Training Durations. In *Advances in Neural Information Processing  
1449 Systems (NeurIPS)*, Vol. 37. Spotlight. Practical scaling laws with constant LR +  
1450 cooldowns for reliable training compute prediction..
- 1451 [20] Amer Haj-Ali et al. 2025. Omniwise: Predicting GPU Kernels Performance with  
1452 LLMs. *arXiv preprint arXiv:2506.20886* (2025). First LLM-based GPU kernel  
1453 performance prediction, 90% within 10% error on AMD MI250/MI300X.
- 1454 [21] Yanbin Hao et al. 2025. POD-Attention: Unlocking Full Prefill-Decode Overlap  
1455 for Faster LLM Inference. In *Proceedings of the 30th ACM International Conference  
1456 on Architectural Support for Programming Languages and Operating Systems (AS-  
1457 PLOS)*. 1–15. Full overlap between prefill and decode phases for LLM inference.
- 1458 [22] John L. Hennessy and David A. Patterson. 2019. A New Golden Age for Computer  
1459 Architecture. *Commun. ACM* 62, 2 (2019), 48–60. <https://doi.org/10.1145/3282307>
- 1460 Turing Award Lecture: domain-specific architectures and the end of Dennard  
1461 scaling.
- 1462 [23] Guseul Heo, Sangyeop Lee, Jaehong Cho, Hyunmin Choi, Sanghyeon Lee,  
1463 Hyungkyu Ham, Gwangsun Kim, Divya Mahajan, and Jongse Park. 2024. Ne-  
1464 uPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing. In  
1465 *Proceedings of the 29th ACM International Conference on Architectural Support  
1466 for Programming Languages and Operating Systems (ASPLOS)*. 1–17. NPU-  
1467 PIM heterogeneous architecture for LLM inference with performance modeling.  
KAIST/Georgia Tech..
- 1468 [24] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya,  
1469 Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes  
1470 Welbl, Aidan Clark, et al. 2022. Training Compute-Optimal Large Language  
1471 Models. *arXiv preprint arXiv:2203.15556* (2022). Chinchilla scaling laws: compute-  
1472 optimal training requires scaling data proportionally to model size.
- 1473 [25] Rodrigo Huerta, Mojtaba Abaie Shoushtary, Jose-Lorenzo Cruz, and Antonio  
1474 Gonzalez. 2025. Dissecting and Modeling the Architecture of Modern GPU Cores.  
In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture  
1475 (MICRO)*. 369–384. Reverse-engineers modern NVIDIA GPU cores, improves  
1476 Accel-Sim to 13.98% MAPE. UPC Barcelona..
- 1477 [26] Bongjoon Hyun, Taehun Kim, Dongjae Lee, and Minsoo Rhu. 2024. Pathfinding  
1478 Future PIM Architectures by Demystifying a Commercial PIM Technology.  
In *Proceedings of the IEEE International Symposium on High Performance Computer  
1479 Architecture (HPCA)*. 1–15. uPIMulator: cycle-accurate PIM simulation  
1480 framework for UMEM. KAIST..
- 1481 [27] Anand Jayaraman, Wei-Lin Hu, Gauri Zhao, and Gennady Pekhimenko. 2023.  
Sia: Heterogeneity-aware, Goodput-optimized ML-Cluster Scheduling. In *Proceedings  
1482 of the 29th Symposium on Operating Systems Principles (SOSP)*. 642–657.  
<https://doi.org/10.1145/3600006.3613175> Extends goodput optimization to het-  
1483 erogeneous GPU clusters for training workloads.
- 1484 [28] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil,  
1485 James Laudon, Cliff Young, and David Patterson. 2023. TPU v4: An Optically  
1486 Reconfigurable Supercomputer for Machine Learning with Hardware Support for  
1487 Embeddings. *Proceedings of the 50th Annual International Symposium on Computer  
1488 Architecture (ISCA)* (2023), 1–14. <https://doi.org/10.1145/3579371.3589350> 4096-  
1489 chip pods with 3D optical interconnect; up to 1.7x/2.1x faster than TPU v3.
- 1490 [29] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal,  
1491 Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borber, et al. 2017.  
In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings  
1492 of the 44th Annual International Symposium on Computer Architecture (ISCA)*. 1–12.  
<https://doi.org/10.1145/3079856.3080246> First dedicated ML inference  
1493 accelerator; 15–30x over CPUs/GPUs on CNN inference.
- 1494 [30] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios  
1495 Kydis, and Dimitrios Soudris. 2025. throttLLM: Predictive GPU Throttling for  
1496 Energy Efficient LLM Inference Serving. In *Proceedings of the IEEE International  
1497 Symposium on High Performance Computer Architecture (HPCA)*. 1–14. Achieves  
1498 up to 43.8% lower energy consumption for LLM inference.
- 1499 [31] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess,  
1500 Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020.  
Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*  
1501 (2020). Original neural scaling laws: power-law relationships between model  
1502 size, dataset size, compute, and loss.
- 1503 [32] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020.  
Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In  
1504 *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*.  
473–486. <https://doi.org/10.1109/ISCA45697.2020.00047>
- 1505 [33] Jungho Kim et al. 2025. PyTorchSim: A Comprehensive, Fast, and Accurate  
1506 NPU Simulation Framework. In *Proceedings of the 58th IEEE/ACM International  
1507 Symposium on Microarchitecture (MICRO)*. 1–14. <https://doi.org/10.1145/3725843.3756045> PyTorch 2-integrated NPU simulator with custom RISC-V ISA and  
1508 Tile-Level Simulation.
- 1509 [34] Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. 2026. The Cost of  
1510 Dynamic Reasoning: Demystifying AI Agents and Test-Time Scaling from an AI  
1511 Infrastructure Perspective. In *Proceedings of the IEEE International Symposium  
1512 on High Performance Computer Architecture (HPCA)*. 1–14. HPCA 2026 (Jan 31–  
1513 Feb 4, 2026, Las Vegas). First comprehensive system-level analysis of AI agents;  
1514 quantifies resources usage, latency, and datacenter power consumption.
- 1515 [35] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Norman P.  
1516 Jouppi, Jignesh Parmar, Hyoukjun Kim, James Laudon, and Chandrakant  
1517 Narayanaswami. 2023. ArchGym: An Open-Source Gymnasium for Machine  
1518 Learning Assisted Architecture Design. In *Proceedings of the 50th International  
1519 Symposium on Computer Architecture (ISCA)*. 1–16. <https://doi.org/10.1145/3579371.3589049>
- 1520 [36] Hyoukjun Kwon, Prasanth Chatarasi, Michael Sarber, Michael Pellauer, Angshuman  
1521 Parashar, and Tushar Krishna. 2019. MAESTRO: A Data-Centric Approach  
1522 to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. In  
1523 *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture  
1524 (MICRO)*. 1–14. <https://doi.org/10.1145/3352460.3358292>
- 1525 [37] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng,  
1526 Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient  
1527 Memory Management for Large Language Model Serving with PagedAttention.  
In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*.  
611–626. <https://doi.org/10.1145/3600006.3613165>

- 1509 [38] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis,  
Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific  
Computation. In *Proceedings of the IEEE/ACM International Symposium on Code  
Generation and Optimization (CGO)*. 2–14. <https://doi.org/10.1109/CGO51591.2021.9370308>
- 1510 Multi-level IR infrastructure enabling cost model composition  
across abstraction levels.
- 1511 [39] Hyojung Lee, Daehyeon Baek, Jimyoung Son, Jieun Choi, Kihyo Moon, and  
Minsung Jang. 2025. PAISE: PIM-Accelerated Inference Scheduling Engine for  
Transformer-based LLM. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 1–14. PIM-based LLM  
inference scheduling. 48.3% speedup, 11.5% power reduction. Samsung..
- 1512 [40] Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. 2021. HELP:  
Hardware-Adaptive Efficient Latency Prediction for NAS via Meta-Learning.  
In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34. 27016–  
27028.
- 1513 [41] Seunghyun Lee, Amar Phanishayee, and Divya Mahajan. 2025. NeuSight: GPU  
Performance Forecasting via Tile-Based Execution Analysis. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming  
Languages and Operating Systems (ASPLOS)*. 1–15.
- 1514 [42] Jianbo Li et al. 2025. TrioSim: A Lightweight Simulator for Large-Scale DNN  
Workloads on Multi-GPU Systems. In *Proceedings of the 52nd Annual International  
Symposium on Computer Architecture (ISCA)*. 1–13. Multi-GPU DNN simulation  
with lightweight approach for distributed training analysis.
- 1515 [43] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020.  
DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109. <https://doi.org/10.1109/LCA.2020.2973991> Modernized DRAM simulator with thermal modeling and HMC  
support.
- 1516 [44] Wenzxian Liang et al. 2025. Lumos: Efficient Performance Modeling and Esti-  
mation for Large-scale LLM Training. In *Proceedings of Machine Learning and  
Systems (MLSys)*. 1–16. Trace-driven performance modeling achieving 3.3% error  
on H100 GPUs for LLM training.
- 1517 [45] Haocong Luo, Yahya Can Tugrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray  
Yaglikci, and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible  
DRAM Simulator. *IEEE Computer Architecture Letters* 22, 2 (2023), 129–132.  
<https://doi.org/10.1109/LCA.2023.3333759> Modular DRAM simulator with DDR5,  
LPDDR5, HBM3, GDDR6 support and RowHammer mitigation modeling.
- 1518 [46] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micike-  
vicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf,  
et al. 2020. MLPerf Training Benchmark. In *Proceedings of Machine Learning and  
Systems (MLSys)*. 336–349. Standard ML training benchmark suite covering  
image classification, object detection, NLP, recommendation, reinforcement  
learning.
- 1519 [47] Azaz Ur-Rehman Nasir, Samroz Ahmad Shoib, Muhammad Abdullah Hanif, and  
Muhammad Shafique. 2025. ESM: A Framework for Building Effective Surrogate  
Models for Hardware-Aware Neural Architecture Search. In *Proceedings of the  
62nd ACM/IEEE Design Automation Conference (DAC)*. 1–6. 97.6% accuracy  
surrogate model framework for HW-aware NAS.
- 1520 [48] Amir Nasr-Esfahany et al. 2025. Concorde: Fast and Accurate CPU Performance  
Modeling with Compositional Analytical-ML Fusion. In *Proceedings of the 52nd  
Annual International Symposium on Computer Architecture (ISCA)*. 1–15. Hybrid  
analytical-ML approach achieving 2% CPI error at 5 orders of magnitude faster  
than gem5.
- 1521 [49] NVIDIA Corporation. 2019. Nsight Compute: Interactive Kernel Profiler. <https://developer.nvidia.com/nsight-compute>. Industry-standard GPU kernel profiling  
tool with roofline analysis.
- 1522 [50] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen,  
Victor A. Ying, Anurag Muber, Rangharajan Venkatesan, Brucek Khailany,  
Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach  
to DNN Accelerator Evaluation. In *Proceedings of the IEEE International Sym-  
posium on Performance Analysis of Systems and Software (ISPASS)*. 304–315.  
<https://doi.org/10.1109/ISPASS.2019.00042>
- 1523 [51] Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk  
Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of  
PIM for Batched Transformer-based Generative Model Inference. In *Proceedings of  
the 29th ACM International Conference on Architectural Support for Programming  
Languages and Operating Systems (ASPLOS)*. 1–16. PIM-based accelerator for  
batched transformer attention. Seoul National University/UIUC..
- 1524 [52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory  
Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019.  
PyTorch: An Imperative Style, High-Performance Deep Learning Library. In  
*Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32. 8024–8035.
- 1525 [53] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aakanksha Shah, Íñigo Goiri,  
Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM  
Inference Using Phase Splitting. In *Proceedings of the 51st Annual International  
Symposium on Computer Architecture (ISCA)*. 118–132. <https://doi.org/10.1109/>
- 1526 [54] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance  
Model for Deep Neural Networks. In *Proceedings of the 5th International Con-  
ference on Learning Representations (ICLR)*. <https://openreview.net/forum?id=SyVVJ85lg>
- 1527 [55] 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.
- 1528 [56] Jonathan Ragan-Kelley, Connally 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> Pioneer separation of algorithm and schedule with learned  
cost models for autoscheduling.
- 1529 [57] Mehdi Rakhsanfar 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>
- 1530 [58] 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>
- 1531 [59] Vijay Janapa Reddi et al. 2025. MLPerf Power: Benchmarking the Energy Effi-  
ciency 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.
- 1532 [60] Vijay Janapa Reddi, Christine Cheng, David Mattson, Guenther Schmuelling,  
Carole-Jean Wu, Brian Anderson, Maxim Breeshnev, 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.
- 1533 [61] 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](https://doi.org/10.1007/978-3-642-12331-3_2)
- 1534 [62] 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.
- 1535 [63] Zhuomin Shen, Jaeho Kim, et al. 2025. AQUA: Network-Accelerated Memory  
Offloading for LLMs in Scale-Up GPU Domains. In *Proceedings of the 30th ACM  
International Conference on Architectural Support for Programming Languages  
and Operating Systems (ASPLOS)*. 1–16. <https://doi.org/10.1145/3676641.3715983>  
Improves LLM inference responsiveness by 20x through network-accelerated  
memory offloading.
- 1536 [64] 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).
- 1537 [65] 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.
- 1538 [66] Ondrej Sykora, Alexis Rucker, Charith Mendis, Rajkishore Barik,  
Phitchaya Mangpo Phothilimthana, and Saman Amarasinghe. 2022. GRANITE:  
A Graph Neural Network Model for Basic Block Throughput Estimation. In  
*Proceedings of the IEEE International Symposium on Workload Characterization  
(ISWC)*. 1–13. <https://doi.org/10.1109/ISWC55918.2022.00014>
- 1539 [67] 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.
- 1540 [68] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: An Intermediate Lan-  
guage and Compiler for Tiled Neural Network Computations. In *Proceedings  
of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Pro-  
gramming Languages (MAPL)*. 10–19. <https://doi.org/10.1145/3315508.3329973>  
Tile-based GPU programming with heuristic performance model for kernel  
generation.

- 1625 [69] Adrian Tschand, Mohamed Awad, et al. 2025. SwizzlePerf: Hardware-Aware  
1626 LLMs for GPU Kernel Performance Optimization. *arXiv preprint arXiv:2508.20258*  
1627 (2025). LLM-based spatial optimization for GPU kernels, up to 2.06x speedup  
1628 via swizzling.
- 1629 [70] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Heyang Zhou, Sen Zhang, Yikai  
1630 Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, et al. 2025. SimAI: Unifying Ar-  
1631 chitecture Design and Performance Tuning for Large-Scale LLM Training with  
1632 Scalability and Precision. In *Proceedings of the 22nd USENIX Symposium on  
1633 Networked Systems Design and Implementation (NSDI)*. 1–18. Full-stack LLM  
1634 training simulator achieving 98.1% alignment with real-world results. Alibaba  
1635 Cloud/Tsinghua..
- 1636 [71] Zixian Wang et al. 2025. SynPerf: Synthesizing High-Performance GPU Kernels  
1637 via Pipeline Decomposition. *arXiv preprint* (2025). Under review.
- 1638 [72] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An  
1639 Insightful Visual Performance Model for Multicore Architectures. *Commun.  
1640 ACM* 52, 4 (2009), 65–76. <https://doi.org/10.1145/1498765.1498785>
- 1641 [73] William Won, Taekyung Heo, Saeed Rashidi, Saeed Talati, Srinivas Srinivasan,  
1642 and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and  
1643 Disaggregated Systems for Large-Model Training at Scale. In *Proceedings of the  
1644 IEEE International Symposium on Performance Analysis of Systems and Software  
1645 (ISPASS)*. 283–294. <https://doi.org/10.1109/ISPASS57527.2023.00035>
- 1646 [74] Yannan Nellie Wu, Joel Emer, and Vivienne Sze. 2022. Sparseloop: An Analytical  
1647 Approach to Sparse Tensor Accelerator Modeling. In *Proceedings of the 55th  
1648 IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–15. <https://doi.org/10.1109/MICRO56248.2022.00078>
- 1649 [75] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-  
1650 Gon Chun. 2022. ORCA: A Distributed Serving System for Transformer-Based  
1651 Generative Models. In *Proceedings of the 16th USENIX Symposium on Operating  
1652 Systems Design and Implementation (OSDI)*. 521–538.
- 1653
- 1654
- 1655
- 1656
- 1657
- 1658
- 1659
- 1660
- 1661
- 1662
- 1663
- 1664
- 1665
- 1666
- 1667
- 1668
- 1669
- 1670
- 1671
- 1672
- 1673
- 1674
- 1675
- 1676
- 1677
- 1678
- 1679
- 1680
- 1681
- 1682
- [76] Geoffrey X. Yu, Yubo Gao, Pavel Golber, and Asaf Cidon. 2021. Habitat: A  
1683 Runtime-Based Computational Performance Predictor for Deep Neural Network  
1684 Training. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 503–  
1685 521.
- [77] Yi Zhai, Yu Cheng Wang, Peng Jiang, and Congming Kang. 2023. TLP: A Deep  
1686 Learning-based Cost Model for Tensor Program Tuning. In *Proceedings of the  
1687 28th ACM International Conference on Architectural Support for Programming  
1688 Languages and Operating Systems (ASPLOS)*. 833–845. <https://doi.org/10.1145/3575693.3575736>
- [78] Li Lyra Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang,  
1689 and Yunxin Liu. 2021. nn-Meter: Towards Accurate Latency Prediction of Deep-  
1690 Learning Model Inference on Diverse Edge Devices. In *Proceedings of the 19th  
1691 Annual International Conference on Mobile Systems, Applications, and Services  
1692 (MobiSys)*. 81–93. <https://doi.org/10.1145/3458864.3467882> Best Paper Award.
- [79] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Amer  
1693 Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez,  
1694 and Ion Stoica. 2020. AnsoR: Generating High-Performance Tensor Programs  
1695 for Deep Learning. In *Proceedings of the 14th USENIX Symposium on Operating  
1696 Systems Design and Implementation (OSDI)*. 863–879.
- [80] Lianmin Zheng, Ruochen Liu, Junru Shao, Tianqi Chen, Joseph E. Gonzalez,  
1697 Ion Stoica, and Zhihao Zhang. 2021. TenSet: A Large-scale Program Perform-  
1698 ance Dataset for Learned Tensor Compilers. In *Advances in Neural Information  
1699 Processing Systems (NeurIPS)*, Vol. 34. 29876–29888.
- [81] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianyu Hu, Yibo Zhu, Xuanzhe Liu,  
1700 Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding  
1701 for Goodput-optimized Large Language Model Serving. In *Proceedings of the 18th  
1702 USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.  
1703 1–18.
- 1704
- 1705
- 1706
- 1707
- 1708
- 1709
- 1710
- 1711
- 1712
- 1713
- 1714
- 1715
- 1716
- 1717
- 1718
- 1719
- 1720
- 1721
- 1722
- 1723
- 1724
- 1725
- 1726
- 1727
- 1728
- 1729
- 1730
- 1731
- 1732
- 1733
- 1734
- 1735
- 1736
- 1737
- 1738
- 1739