

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

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

Abstract

Machine learning-based performance modeling has emerged as a powerful alternative to traditional analytical models and cycle-accurate simulators for predicting computer system behavior. This survey focuses specifically on *ML techniques* for performance prediction across CPUs, GPUs, accelerators, and distributed systems, covering over 60 papers from architecture and ML venues published between 2016–2026. We position traditional analytical models (Timeloop, MAESTRO) and simulators (gem5, GPGPU-Sim) as *baselines* that ML approaches aim to replace or augment. We organize ML approaches along three primary dimensions—modeling technique, target hardware, and input representation—while additionally characterizing papers by workload coverage, prediction targets, accuracy metrics, and evaluation scope. Our analysis reveals that specialized ML models achieve remarkable accuracy—below 5% error for narrow domains—while general-purpose models trade accuracy for broader applicability. Transfer learning and meta-learning techniques increasingly enable adaptation to new hardware with minimal profiling, addressing the challenge of hardware diversity. We identify key open challenges including benchmark diversity, cross-platform generalization, and integration with compiler and architecture exploration workflows. Hybrid approaches combining analytical structure with learned components represent the most promising direction, offering both interpretability and accuracy. This survey provides practitioners guidance for selecting appropriate ML techniques and researchers a roadmap for advancing the field.

Keywords

machine learning, performance modeling, computer architecture, neural networks, survey

1 Introduction

Performance modeling is fundamental to computer architecture research and development. Architects rely on accurate performance predictions to navigate vast design spaces, optimize hardware-software co-design, and make informed decisions about resource allocation. Traditional approaches—analytical models [19] and cycle-accurate simulators [3]—have served the community well, but face growing challenges as workloads and hardware become increasingly complex. Analytical models often oversimplify system behavior, while simulators can require hours or days to evaluate a single design point, making exhaustive exploration impractical.

The rise of deep learning workloads has intensified these challenges. Modern neural networks exhibit diverse computational patterns—from dense matrix operations in transformers to sparse irregular accesses in graph neural networks—that stress traditional

modeling assumptions. Simultaneously, hardware diversity has exploded: GPUs, TPUs, custom accelerators, and multi-device distributed systems each present unique performance characteristics that resist unified analytical treatment. This complexity has motivated a new generation of *machine learning-based* performance models that learn predictive functions directly from profiling data.

ML-based performance modeling has emerged as a compelling alternative. Learned models can capture complex, non-linear relationships between workload characteristics and hardware behavior that elude closed-form analysis. Recent work demonstrates remarkable accuracy: NeuSight [13] achieves 2.3% error predicting GPT-3 latency on H100 GPUs, while nn-Meter [23] reaches 99% accuracy for edge device latency prediction. Beyond accuracy, these approaches offer practical benefits: models trained on one platform can transfer to new hardware with minimal adaptation [7], and inference-time predictions complete in milliseconds rather than hours.

This survey provides a comprehensive analysis of ML-based performance modeling techniques for computer architecture. We focus specifically on *learned* models that acquire predictive capability from data, positioning traditional analytical and simulation approaches as baselines that contextualize ML advances. We make the following contributions:

- A **taxonomy** organizing ML approaches along three primary dimensions (modeling technique, target hardware, input representation), with additional characterization by workload coverage, prediction targets, and accuracy.
- A **systematic survey** of over 60 ML-based performance modeling papers from architecture venues (MICRO, ISCA, HPCA, ASPLOS) and ML venues (MLSys, NeurIPS, ICML) published between 2016–2026.
- A **comparative analysis** examining trade-offs between accuracy, training cost, generalization, and interpretability across ML approaches.
- An identification of **open challenges** including data scarcity, cross-platform generalization, and integration with design automation flows.

The remainder of this paper is organized as follows. Section 2 provides background on traditional performance modeling and relevant ML techniques. Section 3 presents our classification taxonomy. Section 4 surveys approaches organized by target hardware platform. Section 5 offers comparative analysis across key dimensions. Section 6 discusses open challenges and future directions. Section 7 presents hands-on reproducibility evaluations of representative tools. Section 8 concludes.

Figure 1 illustrates the evolution of ML-based performance modeling, showing how techniques have progressed from simple regression models to sophisticated hybrid approaches achieving sub-5% accuracy.

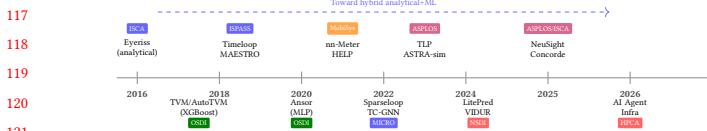


Figure 1: Evolution of ML-based performance modeling (2016–2026). Early work used analytical models (Eyeriss, Timeloop); ML approaches began with simple regressors (TVM) and progressed to deep learning (Ansor, HELP), GNNs (TC-GNN), and transformers (TLP). Current state-of-the-art combines analytical structure with neural networks (NeuSight, Concorde). Recent work extends to AI agent infrastructure analysis.

130

2 Background

2.1 Traditional Performance Modeling

Performance modeling traditionally relies on analytical models and cycle-accurate simulation, which serve as baselines for ML techniques.

2.1.1 Analytical Models. Analytical models express performance as closed-form functions. The roofline model [19] bounds performance by $P = \min(\pi, \beta \cdot I)$, where π is peak FLOPS, β is memory bandwidth, and I is operational intensity. For DNN accelerators, Timeloop [17] models data movement across memory hierarchies, MAESTRO [11] provides data-centric dataflow analysis, and Sparseloop [21] extends to sparse tensors with $2000\times$ speedup over RTL. Analytical models offer fast evaluation (microseconds) and interpretability, but require manual derivation per architecture and struggle with complex microarchitectural effects.

2.1.2 Cycle-Accurate Simulation. Simulators like gem5 [3] for CPUs and GPGPU-Sim [2]/Accel-Sim [8] for GPUs model hardware at register-transfer level, achieving 5–15% accuracy. However, simulating a single ResNet-50 inference may require hours. ASTRA-sim [20] uses analytical abstractions for distributed training but still struggles with the scale of modern workloads.

2.1.3 The Modeling Gap. Analytical models are fast but imprecise; simulators are accurate but slow. ML-based models offer a middle path: learning complex relationships from profiling data while enabling millisecond-scale inference.

2.2 Machine Learning Fundamentals

We briefly review ML techniques used in performance modeling.

Classical ML. Tree-based ensembles (XGBoost, LightGBM) dominate when training data is limited (<10K samples), often outperforming deep learning in low-data regimes [23].

Deep Learning. MLPs learn hierarchical features through stacked transformations. RNNs/LSTMs process sequential inputs but are increasingly replaced by attention-based models.

Graph Neural Networks. GNNs update node representations by aggregating neighbor information: $\mathbf{h}_v^{(k+1)} = \phi(\mathbf{h}_v^{(k)}, \bigoplus_{u \in N(v)} \psi(\mathbf{h}_u^{(k)}, \mathbf{e}_{uv}))$. DNN computation graphs naturally map to this representation, with operators as nodes and data dependencies as edges [18].

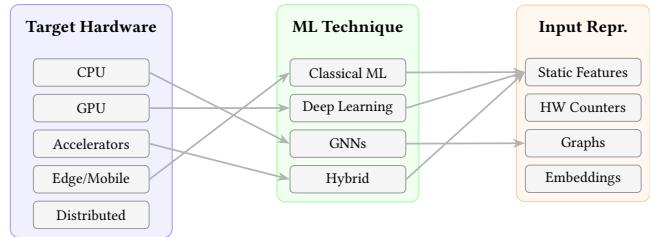


Figure 2: Three-dimensional taxonomy for ML-based performance modeling. Arrows indicate common pairings observed in the literature (e.g., GPU models often use deep learning with static features).

Transformers. Self-attention enables long-range dependency modeling without sequential processing, capturing complex inter-operator interactions.

Transfer Learning. Enables training on easily-profiled hardware and transferring to new platforms with limited data through fine-tuning, domain adaptation, or meta-learning [7].

2.3 Problem Formulation

Performance modeling maps workload \mathcal{W} and hardware \mathcal{H} to metric $y: \hat{y} = f(\mathcal{W}, \mathcal{H}; \theta)$. Workloads are represented at operator-level (layer parameters), graph-level (computation graphs), IR-level (compiler representations), or trace-level (runtime behavior). Hardware is characterized by specifications, performance counters, or learned embeddings.

Prediction targets include latency (execution time), throughput (samples/second), energy (Joules/inference), and memory footprint. Multi-objective formulations enable Pareto-optimal design selection.

Accuracy metrics include MAPE (scale-invariant relative error), RMSE (penalizes large errors), and correlation coefficients (Pearson, Kendall's τ) for ranking accuracy.

Hardware targets span CPUs (cache hierarchies, out-of-order execution), GPUs (SIMT, warp scheduling), accelerators (systolic arrays, dataflows), edge devices (power/memory constraints), and distributed systems (communication, synchronization). This diversity motivates our taxonomy in Section 3.

3 Taxonomy

We organize the literature along three dimensions: hardware target, ML technique, and input representation. Figure 2 illustrates how these dimensions intersect. This framework helps researchers identify explored versus open combinations, and enables practitioners to match methods to their requirements.

3.1 By Modeling Target

The choice of hardware target fundamentally shapes model design, as different platforms exhibit distinct performance characteristics and modeling challenges.

3.1.1 CPU Performance Modeling. CPUs present complex modeling challenges due to deep out-of-order pipelines, sophisticated

cache hierarchies, and branch prediction. ML models for CPU performance must capture instruction-level parallelism, cache behavior, and memory access patterns. Traditional approaches relied on microbenchmark-based linear regression [3], while recent work employs graph neural networks to model basic block throughput [18]. CPU modeling remains challenging due to the diversity of microarchitectures and the difficulty of capturing dynamic effects like branch misprediction and cache contention.

3.1.2 GPU Performance Modeling. GPUs dominate modern ML training and inference, making accurate GPU performance prediction critical. GPU modeling must account for SIMD execution, warp scheduling, memory coalescing, and memory bandwidth limitations. Early approaches used analytical roofline models [19], but these struggle with the complex memory hierarchies and occupancy effects of modern GPUs.

ML-based GPU models have achieved remarkable accuracy. NeuSight [13] introduces tile-based prediction that mirrors CUDA’s execution model, achieving 2.3% error on GPT-3 inference across H100, A100, and V100 GPUs. Habitat [22] pioneered runtime-based cross-GPU prediction using wave scaling analysis. These approaches demonstrate that learned models can capture GPU performance characteristics that elude analytical treatment.

3.1.3 DNN Accelerator Modeling. Custom DNN accelerators—including TPUs, NPUs, and systolic array designs—employ specialized dataflows optimized for matrix operations. Modeling these devices requires understanding the interaction between dataflow, memory hierarchy, and tensor tiling.

Analytical frameworks like Timeloop [17] and MAESTRO [11] provide systematic approaches for accelerator design space exploration. Timeloop models data movement and compute utilization for any valid mapping of operations to hardware, achieving 5–10% accuracy versus RTL simulation at 2000× speedup. MAESTRO offers a data-centric perspective using intuitive dataflow directives. Sparseloop [21] extends these frameworks to sparse tensor operations, critical for efficient transformer inference.

ML-based approaches complement analytical models by learning residual corrections or capturing effects not modeled analytically. ArchGym [10] demonstrates that ML surrogate models can achieve 0.61% RMSE while providing 2000× speedup over simulation, enabling rapid design space exploration for accelerator development.

3.1.4 Edge and Mobile Device Modeling. Edge devices impose strict power, memory, and latency constraints, making accurate prediction essential for deploying ML models on mobile phones, IoT devices, and embedded systems. The diversity of edge hardware—spanning mobile CPUs, mobile GPUs, NPUs, and DSPs—creates significant challenges for cross-platform prediction.

nn-Meter [23] addresses this challenge through kernel-level prediction with adaptive sampling, achieving 99% accuracy across mobile CPUs, GPUs, and Intel VPUs. LitePred [7] extends this work with transfer learning, achieving 99.3% accuracy across 85 edge platforms with less than one hour of adaptation per new device. These results demonstrate that ML models can effectively generalize across the heterogeneous edge hardware landscape.

3.1.5 Distributed System Modeling. Multi-GPU and multi-node systems introduce communication overhead, synchronization barriers, and parallelism strategy choices that fundamentally change performance characteristics. Distributed training performance depends on the interplay between compute, memory bandwidth, and network communication.

ASTRA-sim [20] provides end-to-end distributed training simulation, modeling collective communication algorithms, network topology, and compute-communication overlap. VIDUR [1] focuses specifically on LLM inference serving, capturing the unique characteristics of prefill and decode phases, KV cache management, and request scheduling. These simulation frameworks achieve 5–15% accuracy versus real clusters while enabling exploration of parallelization strategies at scale.

TrioSim [14] provides lightweight multi-GPU simulation for large-scale DNN workloads, complementing ASTRA-sim with faster execution through selective fidelity modeling. Lumos [15] specifically targets LLM training performance prediction through trace-driven modeling, achieving 3.3% error on H100 GPUs by capturing training-specific patterns including gradient accumulation, optimizer states, and activation checkpointing.

3.2 By ML Technique

The choice of ML technique reflects trade-offs between accuracy, data efficiency, interpretability, and generalization capability.

3.2.1 Classical Machine Learning. Tree-based ensembles—random forests and gradient boosted trees (XGBoost, LightGBM)—remain highly effective for performance modeling, particularly in low-data regimes. These methods handle non-linear relationships through recursive partitioning, provide feature importance rankings for interpretability, and require minimal hyperparameter tuning.

Classical ML models dominate when training data is limited (<10K samples) or when features are well-engineered. nn-Meter [23] demonstrates that random forests achieve competitive accuracy with careful kernel-level feature engineering. The ALCOP framework combines XGBoost with analytical pre-training, using analytical model predictions as features to accelerate autotuning convergence.

3.2.2 Deep Learning. Multi-layer perceptrons (MLPs) learn hierarchical feature representations without manual feature engineering. MLPs are widely used as the prediction head in more complex architectures and as standalone models when sufficient training data is available. NeuSight [13] uses MLPs to predict tile-level GPU utilization, learning complex interactions between tile parameters and hardware characteristics.

Recurrent neural networks (RNNs and LSTMs) process sequential inputs, making them suitable for modeling operator sequences in neural network execution. However, sequential processing limits parallelization, and attention-based architectures increasingly replace RNNs for sequence modeling tasks.

3.2.3 Graph Neural Networks. Graph neural networks (GNNs) have emerged as particularly effective for performance modeling because computational graphs have natural graph structure. Nodes represent operators with features (type, parameters, shapes), edges represent data dependencies with features (tensor dimensions, datatypes).

349 GNNs propagate performance-relevant information along these de-
 350 pendencies through message passing.

351 GRANITE [18] applies GNNs to basic block throughput estimation,
 352 learning to predict CPU performance from instruction depen-
 353 dency graphs. For DNN workloads, GNN-based models capture
 354 inter-operator interactions that flat feature representations miss.
 355 The graph structure also enables natural handling of variable-size
 356 networks without padding or truncation.

358 **3.2.4 Hybrid Analytical+ML Models.** Hybrid approaches combine
 359 physics-based analytical models with learned components, achiev-
 360 ing both interpretability and high accuracy. The analytical com-
 361 ponent provides a strong prior based on hardware characteristics,
 362 while the ML component learns residual corrections and complex
 363 interactions.

364 This design philosophy has produced state-of-the-art results.
 365 Analytical pre-training initializes ML models with reasonable pre-
 366 dictions, reducing data requirements and improving convergence.
 367 Physics-informed architectures incorporate analytical insights into
 368 model structure—NeuSight’s tile-based prediction mirrors CUDA’s
 369 execution model, providing inductive bias that improves general-
 370 ization. Residual learning trains ML models to predict the error of
 371 analytical models, combining analytical interpretability with ML’s
 372 ability to capture unmodeled effects.

373 The latency predictor study [6] demonstrates that hybrid ap-
 374 proaches with transfer learning achieve 22.5% average improve-
 375 ment over baselines, with up to 87.6% improvement on challenging
 376 cross-platform prediction tasks.

3.3 By Input Representation

381 Input representation determines what information the model can
 382 access and how effectively it can learn performance-relevant pat-
 383 terns.

386 **3.3.1 Static Features.** Static features derive from workload and
 387 hardware specifications without runtime measurement. For DNN
 388 workloads, these include layer parameters (kernel size, channels,
 389 stride, batch size), tensor dimensions, and operator types. Hardware
 390 specifications include core counts, memory sizes, bandwidth, and
 391 clock frequencies.

392 Static features enable prediction without profiling, supporting
 393 use cases like neural architecture search where thousands of can-
 394 didate networks must be evaluated. Feature engineering plays
 395 a critical role: effective representations capture computation-to-
 396 communication ratios, memory footprint estimates, and paralleliza-
 397 tion potential.

400 **3.3.2 Hardware Counters.** Performance counters provide runtime
 401 measurements of hardware behavior: cache miss rates, memory
 402 bandwidth utilization, instruction throughput, and stall cycles. Counter-
 403 based models can capture dynamic effects invisible to static analysis,
 404 including contention, thermal throttling, and runtime scheduling
 405 decisions.

407 The primary limitation is that counter-based models require hard-
 408 ware execution, limiting their applicability for design space explo-
 409 ration or new architecture evaluation. However, for optimizing ex-
 410 isting deployments or debugging performance anomalies, counter-
 411 based models provide valuable insights that static approaches can-
 412 not match.

413 **3.3.3 Graph Representations.** Graph representations encode com-
 414 putational graphs with nodes representing operators and edges
 415 representing data dependencies. Node features capture operator
 416 characteristics (type, parameters), while edge features encode ten-
 417 sor properties (shape, datatype, memory format).

418 Graph representations provide several advantages over flat fea-
 419 ture vectors: they naturally handle variable-size networks, pre-
 420 serve structural information about operator interactions, and enable
 421 permutation-invariant predictions. GNNs operating on these repre-
 422 sentations can learn which subgraph patterns indicate performance
 423 bottlenecks.

425 **3.3.4 Learned Embeddings.** Learned embeddings compress high-
 426 dimensional or categorical information into dense vector represen-
 427 tations. Hardware embeddings represent diverse devices as points
 428 in a learned feature space, enabling transfer learning across plat-
 429 forms. Operator embeddings capture semantic similarities between
 430 operator types that may share performance characteristics.

431 HELP formulates hardware prediction as meta-learning, learning
 432 hardware embeddings that represent devices as black-box functions.
 433 With just 10 measurement samples on a new device, HELP achieves
 434 accurate predictions by positioning the device appropriately in the
 435 learned embedding space. This approach is particularly valuable
 436 for the fragmented edge hardware landscape, where collecting
 437 exhaustive training data for each device is impractical.

438 The comprehensive survey in Section 4 and Table 1 illustrate the
 439 diversity of approaches across these taxonomy dimensions.

4 Survey of Approaches

440 This section surveys ML-based performance modeling approaches
 441 organized by target hardware platform. For each category, we ex-
 442 amine the modeling challenges specific to that platform, describe
 443 representative techniques, and synthesize key findings across the
 444 literature. Table 1 provides a comprehensive comparison of the
 445 surveyed approaches.

4.1 CPU Performance Modeling

446 CPU performance modeling faces challenges from out-of-order ex-
 447 ecution, branch prediction, and deep cache hierarchies. Traditional
 448 cycle-accurate simulation via gem5 [3] achieves 10–20% accuracy
 449 but requires hours per DNN inference.

450 GRANITE [18] uses GNNs to predict basic block throughput
 451 from instruction dependency graphs, achieving 0.97 Kendall’s τ
 452 on x86 and generalizing across microarchitectures. Concorde [16]
 453 advances hybrid modeling, achieving 2% CPI error at five orders of
 454 magnitude faster than gem5 through compositional analytical-ML
 455 fusion. Key remaining challenges include memory-bound execution
 456 (GRANITE focuses on compute-bound blocks), complex prefetchers,
 457 and optimized library code behavior.

465 **Table 1: Summary of surveyed ML-based performance modeling approaches, organized by target hardware platform.**
 466 *PyTorchSim and TrioSim focus on simulation speedup rather than reporting absolute accuracy vs. real hardware; accu-
 467 racy data not comparable to other entries.

469	Paper	Platform	ML Technique	Prediction Target	Error	Key Innovation	527	
<i>CPU Performance Modeling</i>								
471	GRANITE [18]	CPU	GNN	Basic block throughput	0.97 corr	Instruction graph encoding	529	
472	Concorde [16]	CPU	Hybrid	CPI	2%	Compositional analytical-ML	530	
473	gem5+ML [3]	CPU	Hybrid	Execution time	10–20%	Simulation + learning	531	
474	<i>GPU Performance Modeling</i>							
475	NeuSight [13]	GPU	Hybrid MLP	Kernel/E2E latency	2.3%	Tile-based prediction	533	
476	AMALI [4]	GPU	Analytical	LLM inference	23.6%	Memory hierarchy modeling	534	
477	Habitat [22]	GPU	MLP	Training time	11.8%	Wave scaling analysis	535	
478	Accel-Sim [8]	GPU	Simulation	Cycle-accurate	10–20%	SASS trace-driven	536	
479	<i>DNN Accelerator Modeling</i>							
480	Timeloop [17]	NPU	Analytical	Latency/Energy	5–10%	Loop-nest DSE	538	
481	MAESTRO [11]	NPU	Analytical	Latency/Energy	5–15%	Data-centric directives	539	
482	Sparseloop [21]	NPU	Analytical	Sparse tensors	5–10%	Compression modeling	540	
483	PyTorchSim [9]	NPU	Simulation	Latency	N/A*	PyTorch 2 integration	541	
484	ArchGym [10]	Multi	RL+Surrogate	Multi-objective	0.61%	ML-aided DSE	542	
485	<i>Edge Device Modeling</i>							
486	nn-Meter [23]	Edge	RF ensemble	Latency	<1%	Kernel detection	544	
487	LitePred [7]	Edge	VAE+MLP	Latency	0.7%	85-platform transfer	545	
488	HELP [12]	Multi	Meta-learning	Latency	1.9%	10-sample adaptation	546	
489	<i>Distributed and LLM Systems</i>							
490	ASTRA-sim [20]	Distributed	Simulation	Training time	5–15%	Collective modeling	548	
491	TrioSim [14]	Multi-GPU	Simulation	DNN training	N/A*	Lightweight multi-GPU	549	
492	Lumos [15]	Distributed	Trace-driven	LLM training	3.3%	H100 training modeling	550	
493	VIDUR [1]	GPU cluster	Simulation	LLM serving	<5%	Prefill/decode phases	551	

4.2 GPU Performance Modeling

GPU modeling is challenging due to SIMD execution, complex memory hierarchies, and workload-dependent scheduling. Cycle-accurate simulators (GPGPU-Sim [2], Accel-Sim [8]) achieve 0.90–0.97 IPC correlation but suffer 1000–10000× slowdown.

Learned models. Habitat [22] introduced wave scaling, decomposing execution into compute and memory components that scale with hardware parameters, achieving 11.8% error across GPU generations. NeuSight [13] advances this with tile-based prediction mirroring CUDA execution semantics, achieving 2.3% error on GPT-3 inference (H100, A100, V100)—a 50× improvement over Habitat.

Compiler cost models. TVM [5] and Anstor [24] use XGBoost/MLP models to guide autotuning, achieving ~15% MAPE. The TenSet dataset (52M records) enables pre-trained models that accelerate autotuning 10×.

LLM inference. LLM execution exhibits distinct prefill (compute-bound) and decode (memory-bound) phases. VIDUR [1] provides discrete-event simulation for serving systems, achieving <5% error on end-to-end metrics. AMALI [4] reduces analytical modeling MAPE from 127% to 24% through improved memory hierarchy modeling.

4.3 Accelerator Performance Modeling

DNN accelerators employ specialized dataflows and memory hierarchies optimized for tensor operations.

Analytical modeling. Timeloop [17] analytically computes data reuse, latency, and energy from loop-nest representations, achieving 5–10% accuracy with 2000× speedup over RTL. MAESTRO [11] offers data-centric dataflow directives; Sparseloop [21] extends to sparse tensors.

ML-augmented design. ArchGym [10] connects ML optimization algorithms to simulators, with surrogate models achieving 0.61% RMSE at 2000× speedup. PyTorchSim [9] integrates PyTorch 2 with NPU simulation supporting custom RISC-V ISA and systolic arrays.

Emerging accelerators (PIM, neuromorphic, analog) remain underexplored, with fundamentally different characteristics that existing frameworks do not address.

4.4 Memory System Modeling

Memory increasingly dominates ML performance. For DNN workloads, Timeloop [17] computes exact access counts through data reuse analysis. KV cache management has emerged as the dominant LLM serving challenge; vLLM’s PagedAttention achieves 2–4× throughput through virtual memory techniques, while VIDUR [1] models cache allocation and eviction at the system level.

For distributed training, ASTRA-sim [20] simulates collective communication algorithms, network topology, and compute-communication overlap, achieving 5–15% error and enabling parallelization strategy exploration.

581 4.5 Cross-Platform and Transfer Learning

582 Hardware diversity makes per-device training impractical. HELP [12] 639
 583 formulates cross-hardware prediction as meta-learning, achieving 640
 584 93.2% accuracy with just 10 samples on new devices via MAML- 641
 585 style adaptation. LitePred [7] scales to 85 edge devices using VAE- 642
 586 based intelligent sampling, achieving 99.3% accuracy with under 643
 587 one hour of profiling per device. Systematic evaluation [6] shows 644
 588 transfer learning provides 22.5% average improvement, up to 87.6% 645
 589 on challenging transfers. 646
 590

591 Hybrid approaches like SynPerf combine analytical decomposition 647
 592 with learned components, achieving 6.1% kernel-level error. 648
 593 Key open challenges include transformer/MoE transfer (current 649
 594 work focuses on CNNs), cross-workload-type generalization, and 650
 595 continual learning for evolving software stacks 651

596 5 Comparison and Analysis

597 We now analyze trade-offs in accuracy, training cost, generalization, 652
 598 and interpretability. Table 2 summarizes key dimensions. 653
 599

600 *Note: Accuracy figures are reported as stated in original papers. Direct 654
 601 comparison is limited by differences in benchmarks, workloads, 655
 602 hardware targets, and evaluation protocols.*

604 5.1 Accuracy vs. Training Cost

605 Data collection cost varies dramatically: profiling-based methods 656
 606 (nn-Meter) require ~1,000 samples/kernel; transfer learning (HELP, 657
 607 LitePred) reduces this to 10–100 samples. Simulation-based training 658
 608 (ArchGym on Timeloop) avoids hardware entirely. 659

609 We observe three accuracy tiers: (1) Specialized models (<5% error): 660
 610 nn-Meter, NeuSight, LitePred on narrow domains; (2) General- 661
 611 purpose (5–15%): Habitat, Timeloop, MAESTRO; (3) Compiler cost 662
 612 models (15–25%): TVM/AutoTVM, prioritizing ranking over 663
 613 absolute accuracy. Accuracy requirements are use-case dependent—NAS 664
 614 tolerates 10–15% error while hardware procurement demands <5%. 665

617 5.2 Generalization Capabilities

618 **Workload generalization.** GNN-based approaches (GRANITE [18]) 666
 619 generalize via compositional learning, but cross-workload-type 667
 620 transfer (CNN→transformer) remains challenging. NeuSight [13] 668
 621 addresses this through diverse operator training. 669

622 **Hardware generalization.** Three approaches show promise: 670
 623 meta-learning (HELP [12] with hardware embeddings), feature- 671
 624 based transfer (LitePred [7] achieving 92.1% zero-shot accuracy), 672
 625 and analytical decomposition (Habitat [22] separating compute/memory 673
 626 components).

627 **Temporal generalization** as software evolves remains underex- 674
 628 plored; continual learning for evolving stacks is an open direction. 675

631 5.3 Interpretability

632 Analytical models (Timeloop, MAESTRO) provide full interpretabil- 689
 633 ity with actionable bottleneck identification. Classical ML offers 690
 634 feature importance; deep learning is largely opaque. Hybrid 691
 635 approaches (NeuSight, VIDUR) balance interpretability and accuracy 692
 636 by combining analytical structure with learned residuals, enabling 693
 637 “what-if” analysis. 694

638 6 Open Challenges and Future Directions

639 6.1 Data Availability and Quality

640 Existing datasets predominantly cover CNNs; transformers, MoE, 641
 641 and diffusion models remain underrepresented. Hardware diversity 642
 642 creates bottlenecks—LitePred covers 85 devices but the landscape 643
 643 spans hundreds. Measurement noise from thermal throttling and OS 644
 644 scheduling affects reliability; standardized protocols would improve 645
 645 comparability. 646

647 6.2 Model Generalization

648 Cross-workload generalization (CNN→transformer) fails due to 650
 649 different computational patterns. Cross-hardware transfer [7, 12] 651
 650 shows promise for related platforms but cross-family prediction 652
 651 (GPU→TPU) remains elusive. Distribution shift from software evo- 653
 652 lution invalidates models; continual learning is underexplored. 654

655 6.3 Integration with Design Flows

656 Compiler integration (TVM, Ansor) needs uncertainty quantifica- 657
 657 tion to improve exploration-exploitation trade-offs. Architecture 658
 658 exploration (ArchGym) requires active learning for sample effi- 659
 659 ciency. LLM serving needs real-time prediction within microsec- 660
 660 onds; VIDUR provides offline simulation but online adaptation is 661
 661 challenging. 662

663 6.4 Research Opportunities

664 Five high-priority gaps: (1) **Transformer cross-platform trans- 665
 665 fer**—current work focuses on CNNs; attention-based models have 666
 666 distinct memory patterns. (2) **Uncertainty-aware autotuning**— 667
 667 calibrated confidence intervals could reduce measured evaluations. 668
 668 (3) **Dynamic shape/sparse prediction**—existing models assume 669
 669 fixed inputs; LLM serving sees 128–128K tokens. (4) **Unified energy- 670
 670 latency-memory prediction**—ML approaches focus on latency 671
 671 while edge/datacenter need energy. (5) **Temporal robustness 672
 672 benchmarks**—no systematic evaluation of robustness to software 673
 673 evolution. 674

675 6.5 Future Work: Toward Unified Tooling

676 No single tool addresses all needs; fragmentation forces practitioners 677
 677 to manage incompatible dependencies. Key lessons from our 678
 678 evaluation: Docker-first deployment improves reproducibility; hy- 679
 679 brid approaches combining analytical structure with learned com- 680
 680 ponents show best accuracy; portable model formats (ONNX) pre- 681
 681 vent versioning issues. Future directions include unified workload 682
 682 representations and composable modeling engines with container 683
 683 isolation. 684

685 7 Experimental Evaluation

686 We conducted hands-on reproducibility evaluations of five represen- 689
 687 tative tools using a 10-point rubric: Setup (3 pts: Docker availability, 690
 688 clean installation, quick start), Reproducibility (4 pts: reference 691
 689 outputs, determinism, examples), and Usability (3 pts: API clarity, 692
 690 interpretability, maintenance). Table 3 summarizes results. 693

694 **Key findings.** Docker-first tools (Timeloop, ASTRA-sim, VIDUR) 694
 695 scored 8.5+ by isolating dependencies; we executed all three on 695

Table 2: Comparative analysis of representative performance models—including ML-based and analytical/simulation approaches—across key dimensions. The Accuracy column reports the metric and value as given in each original work (e.g., MAPE, RMSE, Kendall’s τ , ranges).

Model	Accuracy (as reported)	Training Data	Adaptation Cost	Generalization	Interpretability	Inference Time
<i>Classical ML</i>						759
nn-Meter [23]	<1% MAPE	1K/kernel	Hours/device	Device-specific	Medium	760 761
XGBoost (TVM) [5]	20% MAPE	10K+	Online	Operator-level	Medium	762 757
<i>Deep Learning</i>						758
NeuSight [13]	2.3% MAPE	100K+	Pre-trained	Cross-GPU	Low	763 764
Habitat [22]	11.8% MAPE	Online profiling runs	None (requires GPU)	Cross-GPU	Medium	765 766
<i>Graph Neural Networks</i>						767
GRANITE [18]	0.97 τ	10K+	Hours	Cross- μ arch	Low	768 769
HELP [12]	1.9% MAPE	Meta-training	10 samples	Cross-platform	Low	766 770
<i>Transfer Learning</i>						771
LitePred [7]	0.7% MAPE	85 platforms	100 samples	85+ devices	Low	772 773
<i>Hybrid Analytical+ML</i>						774
Timeloop [17]	5–10%	Arch spec	None	Any accelerator	High	775 μs
ArchGym [10]	0.61% RMSE	Simulation	Surrogate training	Architecture-specific	Medium	776 ms
VIDUR [1]	<5%	Kernel profiles	Per-model	LLM-specific	High	777 ms

Table 3: Reproducibility evaluation scores (10-point rubric).

Tool	Setup	Reprod.	Usability	Total
Timeloop	3	4	2	9/10
ASTRA-sim	2.5	3	3	8.5/10
VIDUR	2.5	3.5	3	9/10
nn-Meter	2	0	1	3/10
NeuSight	2	3	2.5	7.5/10

both x86_64 and aarch64 without issues. Timeloop provides reference outputs for all examples (Eyeriss, Simba) with deterministic results. ASTRA-sim includes validated HGX-H100 configurations; VIDUR enables scheduler comparison (vLLM, Orca, Sarathi) without GPU hardware. NeuSight’s tile-based hybrid approach achieves 2.3% error on LLM workloads.

Critical anti-pattern. nn-Meter’s pre-trained predictors fail with current scikit-learn due to pickle format changes—a cautionary example of ML model serialization fragility. Projects should prefer portable formats (ONNX) or pin exact dependency versions.

Best practices: (1) Provide Docker images to isolate dependencies; (2) Document Python version requirements; (3) Include reference outputs for validation; (4) Use portable model formats; (5) Pin dependency versions.

8 Conclusion

This survey analyzed over 60 papers on ML-based performance modeling for computer architecture.

Key findings: (1) ML models achieve <5% error on target domains (NeuSight 2.3%, LitePred 0.7%). (2) Hybrid analytical+ML approaches dominate, combining interpretable structure with learned residuals (Concorde, AMALI, Lumos). (3) Transfer learning enables 10–100 sample adaptation via meta-learning (HELP) and VAE sampling (LitePred). (4) Kernel-level decomposition (nn-Meter) enables

compositional predictions. (5) LLM serving requires specialized modeling for prefill/decode phases and KV cache (VIDUR).

Promising directions: Foundation models for performance prediction, uncertainty quantification for autotuning, temporal generalization via continual learning, multi-objective (latency/energy/memory) prediction, and emerging hardware (PIM, neuromorphic) support.

Machine learning has transformed performance modeling into a systematic discipline. As ML workloads grow and hardware diversifies, accurate, generalizable models become critical for efficient system design. This survey serves as both a reference for practitioners and a roadmap for researchers.

References

- [1] 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.
- [2] 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>
- [3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 Simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7. <https://doi.org/10.1145/2024716.2024718>
- [4] Zheng Cao et al. 2025. AMALI: An Analytical Model for Accurately Modeling LLM Inference on Modern GPUs. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. 1–14. <https://doi.org/10.1145/3695053.3731064> Reduces GPU LLM inference MAPE from 127.56% to 23.59% vs GCoM baseline.
- [5] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 578–594.
- [6] Lukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Roysen Lee, Hyeji Kim, and Nicholas D. Lane. 2024. Latency Predictors for Neural Architecture Search. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–14.
- [7] Yang Feng, Zhehao Li, Jiacheng Yang, and Yunxin Liu. 2024. LitePred: Transferable and Scalable Latency Prediction for Hardware-Aware Neural Architecture

- 813 Search. In *Proceedings of the 21st USENIX Symposium on Networked Systems
814 Design and Implementation (NSDI)*. 1–18.
- 815 [8] Mahmoud Khairy, Zheseng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020.
816 Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In
817 *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*.
818 473–486. <https://doi.org/10.1109/ISCA45697.2020.00047>
- 819 [9] Jungho Kim et al. 2025. PyTorchSim: A Comprehensive, Fast, and Accurate
820 NPU Simulation Framework. In *Proceedings of the 58th IEEE/ACM International
821 Symposium on Microarchitecture (MICRO)*. 1–14. <https://doi.org/10.1145/3725843.3756045> PyTorch 2-integrated NPU simulator with custom RISC-V ISA and
822 Tile-Level Simulation.
- 823 [10] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Norman P.
824 Jouppi, Jignesh Parmar, Hyoukjun Kim, James Laudon, and Chandrakant
825 Narayanaswami. 2023. ArchGym: An Open-Source Gymnasium for Machine
826 Learning Assisted Architecture Design. In *Proceedings of the 50th International
827 Symposium on Computer Architecture (ISCA)*. 1–16. <https://doi.org/10.1145/3579371.3589049>
- 828 [11] Hyoukjun Kwon, Prasanth Chatarasi, Michael Sarber, Michael Pellauer, Angshuman
829 Parashar, and Tushar Krishna. 2019. MAESTRO: A Data-Centric Approach
830 to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. In
831 *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture
832 (MICRO)*. 1–14. <https://doi.org/10.1145/3352460.3358292>
- 833 [12] Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. 2021. HELP:
834 Hardware-Adaptive Efficient Latency Prediction for NAS via Meta-Learning.
835 In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34. 27016–
836 27028.
- 837 [13] Seunghyun Lee, Amar Phanishayee, and Divya Mahajan. 2025. NeuSight: GPU
838 Performance Forecasting via Tile-Based Execution Analysis. In *Proceedings of
839 the 30th ACM International Conference on Architectural Support for Programming
840 Languages and Operating Systems (ASPLOS)*. 1–15.
- 841 [14] Jianbo Li et al. 2025. TrioSim: A Lightweight Simulator for Large-Scale DNN
842 Workloads on Multi-GPU Systems. In *Proceedings of the 52nd Annual International
843 Symposium on Computer Architecture (ISCA)*. 1–13. Multi-GPU DNN simulation
844 with lightweight approach for distributed training analysis.
- 845 [15] Wenzuan Liang et al. 2025. Lumos: Efficient Performance Modeling and Esti-
846 mation for Large-scale LLM Training. In *Proceedings of Machine Learning and
847 Systems (MLSys)*. 1–16. Trace-driven performance modeling achieving 3.3% error
848 on H100 GPUs for LLM training.
- 849 [16] Amir Nasr-Esfahany et al. 2025. Concorde: Fast and Accurate CPU Performance
850 Modeling with Compositional Analytical-ML Fusion. In *Proceedings of the 52nd
851 Annual International Symposium on Computer Architecture (ISCA)*. 1–15. Hybrid
852 analytical-ML approach achieving 2% CPI error at 5 orders of magnitude faster
853 than gem5.
- 854 [17] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen,
855 Victor A. Ying, Anurag Muber, Rangharajan Venkatesan, Brucek Khailany,
856 Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach
857 to DNN Accelerator Evaluation. In *Proceedings of the IEEE International Sym-
858 posium on Performance Analysis of Systems and Software (ISPASS)*. 304–315.
<https://doi.org/10.1109/ISPASS.2019.00042>
- 859 [18] Ondrej Sykora, Alexis Rucker, Charith Mendis, Rajkishore Barik,
860 Phitcaya Mangpo Phothilimthana, and Saman Amarasinghe. 2022. GRANITE:
861 A Graph Neural Network Model for Basic Block Throughput Estimation. In
862 *Proceedings of the IEEE International Symposium on Workload Characterization
863 (IISWC)*. 1–13. <https://doi.org/10.1109/IISWC55918.2022.00014>
- 864 [19] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An
865 Insightful Visual Performance Model for Multicore Architectures. *Commun.
866 ACM* 52, 4 (2009), 65–76. <https://doi.org/10.1145/1498765.1498785>
- 867 [20] William Won, Taekyung Heo, Saeed Rashidi, Saeed Talati, Srinivas Srinivasan,
868 and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and
869 Disaggregated Systems for Large-Model Training at Scale. In *Proceedings of the
870 IEEE International Symposium on Performance Analysis of Systems and Software
871 (ISPASS)*. 283–294. <https://doi.org/10.1109/ISPASS57527.2023.00035>
- 872 [21] Yannan Nellie Wu, Joel Emer, and Vivienne Sze. 2022. Sparseloop: An Analytical
873 Approach to Sparse Tensor Accelerator Modeling. In *Proceedings of the 55th
874 IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–15. <https://doi.org/10.1109/MICRO56248.2022.00078>
- 875 [22] Geoffrey X. Yu, Yubo Gao, Pavel Golber, and Asaf Cidon. 2021. Habitat: A
876 Runtime-Based Computational Performance Predictor for Deep Neural Network
877 Training. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 503–
878 521.
- 879 [23] Li Lyra Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang,
880 and Yunxin Liu. 2021. nn-Meter: Towards Accurate Latency Prediction of Deep-
881 Learning Model Inference on Diverse Edge Devices. In *Proceedings of the 19th
882 Annual International Conference on Mobile Systems, Applications, and Services
883 (MobiSys)*. 81–93. <https://doi.org/10.1145/3458864.3467882> Best Paper Award.
- 884 [24] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer
885 Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez,
886 and Ion Stoica. 2020. Ansor: Generating High-Performance Tensor Programs
887 for Deep Learning. In *Proceedings of the 14th USENIX Symposium on Operating
888 Systems Design and Implementation (OSDI)*. 863–879.

871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928