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.

MICRO 2026, Austin, TX, USA
2026. ACM ISBN 978-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

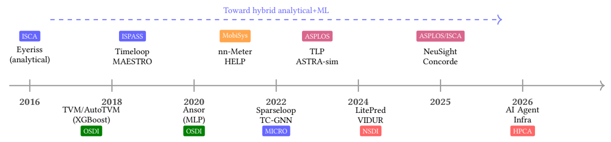

Figure 1: Evolution of ML-based performance modeling (2016–2026). Early work used analytical models (Eyeriss, Timeloop); ML approaches began with simple regressors (TVM) and progressed to deep learning (Ansor, HELP), GNNs (TC-GNN), and transformers (TLP). Current state-of-the-art combines analytical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

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× 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: $\mathbf{h}_v^{(k+1)} = \phi(\mathbf{h}_v^{(k)}, \bigoplus_{u \in \mathcal{N}(v)} \psi(\mathbf{h}_u^{(k)}, \mathbf{e}_{uv}))$. DNN computation graphs naturally map to this representation, with operators as nodes and data dependencies as edges [18].

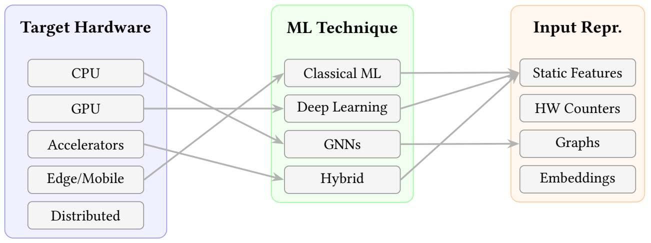

Figure 2: Three-dimensional taxonomy for ML-based performance modeling. Arrows indicate common pairings observed in the literature (e.g., GPU models often use deep learning with static features).

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 SIMT 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).

GNNs propagate performance-relevant information along these dependencies through message passing.

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

3.2.4 Hybrid Analytical+ML Models. Hybrid approaches combine physics-based analytical models with learned components, achieving both interpretability and high accuracy. The analytical component provides a strong prior based on hardware characteristics, while the ML component learns residual corrections and complex interactions.

This design philosophy has produced state-of-the-art results. Analytical pre-training initializes ML models with reasonable predictions, reducing data requirements and improving convergence. Physics-informed architectures incorporate analytical insights into model structure—NeuSight's tile-based prediction mirrors CUDA's execution model, providing inductive bias that improves generalization. Residual learning trains ML models to predict the error of analytical models, combining analytical interpretability with ML's ability to capture unmodeled effects.

The latency predictor study [6] demonstrates that hybrid approaches with transfer learning achieve 22.5% average improvement over baselines, with up to 87.6% improvement on challenging cross-platform prediction tasks.

3.3 By Input Representation

Input representation determines what information the model can access and how effectively it can learn performance-relevant patterns.

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

Static features enable prediction without profiling, supporting use cases like neural architecture search where thousands of candidate networks must be evaluated. Feature engineering plays a critical role: effective representations capture computation-to-communication ratios, memory footprint estimates, and parallelization potential.

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

The primary limitation is that counter-based models require hardware execution, limiting their applicability for design space exploration or new architecture evaluation. However, for optimizing existing deployments or debugging performance anomalies, counter-based models provide valuable insights that static approaches cannot match.

3.3.3 Graph Representations. Graph representations encode computational graphs with nodes representing operators and edges representing data dependencies. Node features capture operator characteristics (type, parameters), while edge features encode tensor properties (shape, datatype, memory format).

Graph representations provide several advantages over flat feature vectors: they naturally handle variable-size networks, preserve structural information about operator interactions, and enable permutation-invariant predictions. GNNs operating on these representations can learn which subgraph patterns indicate performance bottlenecks.

3.3.4 Learned Embeddings. Learned embeddings compress high-dimensional or categorical information into dense vector representations. Hardware embeddings represent diverse devices as points in a learned feature space, enabling transfer learning across platforms. Operator embeddings capture semantic similarities between operator types that may share performance characteristics.

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

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

4 Survey of Approaches

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

4.1 CPU Performance Modeling

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

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

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

Paper	Platform	ML Technique	Prediction Target	Error	Key Innovation
CPU Performance Modeling					
GRANITE [18]	CPU	GNN	Basic block throughput	0.97 corr	Instruction graph encoding
Concorde [16]	CPU	Hybrid	CPI	2%	Compositional analytical-ML
gem5+ML [3]	CPU	Hybrid	Execution time	10–20%	Simulation + learning
GPU Performance Modeling					
NeuSight [13]	GPU	Hybrid MLP	Kernel/E2E latency	2.3%	Tile-based prediction
AMALI [4]	GPU	Analytical	LLM inference	23.6%	Memory hierarchy modeling
Habitat [22]	GPU	MLP	Training time	11.8%	Wave scaling analysis
Accel-Sim [8]	GPU	Simulation	Cycle-accurate	10–20%	SASS trace-driven
DNN Accelerator Modeling					
Timeloop [17]	NPU	Analytical	Latency/Energy	5–10%	Loop-nest DSE
MAESTRO [11]	NPU	Analytical	Latency/Energy	5–15%	Data-centric directives
Sparseloop [21]	NPU	Analytical	Sparse tensors	5–10%	Compression modeling
PyTorchSim [9]	NPU	Simulation	Latency	N/A*	PyTorch 2 integration
ArchGym [10]	Multi	RL+Surrogate	Multi-objective	0.61%	ML-aided DSE
Edge Device Modeling					
nn-Meter [23]	Edge	RF ensemble	Latency	<1%	Kernel detection
LitePred [7]	Edge	VAE+MLP	Latency	0.7%	85-platform transfer
HELP [12]	Multi	Meta-learning	Latency	1.9%	10-sample adaptation
Distributed and LLM Systems					
ASTRA-sim [20]	Distributed	Simulation	Training time	5–15%	Collective modeling
TrioSim [14]	Multi-GPU	Simulation	DNN training	N/A*	Lightweight multi-GPU
Lumos [15]	Distributed	Trace-driven	LLM training	3.3%	H100 training modeling
VIDUR [1]	GPU cluster	Simulation	LLM serving	<5%	Prefill/decode phases

4.2 GPU Performance Modeling

GPU modeling is challenging due to SIMT 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 Ansor [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.

4.5 Cross-Platform and Transfer Learning

Hardware diversity makes per-device training impractical. HELP [12] formulates cross-hardware prediction as meta-learning, achieving 93.2% accuracy with just 10 samples on new devices via MAML-style adaptation. LitePred [7] scales to 85 edge devices using VAE-based intelligent sampling, achieving 99.3% accuracy with under one hour of profiling per device. Systematic evaluation [6] shows transfer learning provides 22.5% average improvement, up to 87.6% on challenging transfers.

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

5 Comparison and Analysis

We now analyze trade-offs in accuracy, training cost, generalization, and interpretability. Table 2 summarizes key dimensions.
Note: Accuracy figures are reported as stated in original papers. Direct comparison is limited by differences in benchmarks, workloads, hardware targets, and evaluation protocols.

5.1 Accuracy vs. Training Cost

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

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

5.2 Generalization Capabilities

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

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

Temporal generalization as software evolves remains underexplored; continual learning for evolving stacks is an open direction.

5.3 Interpretability

Analytical models (Timeloop, MAESTRO) provide full interpretability with actionable bottleneck identification. Classical ML offers feature importance; deep learning is largely opaque. Hybrid approaches (NeuSight, VIDUR) balance interpretability and accuracy by combining analytical structure with learned residuals, enabling "what-if" analysis.

6 Open Challenges and Future Directions

6.1 Data Availability and Quality

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

6.2 Model Generalization

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

6.3 Integration with Design Flows

Compiler integration (TVM, Ansor) needs uncertainty quantification to improve exploration-exploitation trade-offs. Architecture exploration (ArchGym) requires active learning for sample efficiency. LLM serving needs real-time prediction within microseconds; VIDUR provides offline simulation but online adaptation is challenging.

6.4 Research Opportunities

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

6.5 Future Work: Toward Unified Tooling

No single tool addresses all needs; fragmentation forces practitioners to manage incompatible dependencies. Key lessons from our evaluation: Docker-first deployment improves reproducibility; hybrid approaches combining analytical structure with learned components show best accuracy; portable model formats (ONNX) prevent versioning issues. Future directions include unified workload representations and composable modeling engines with container isolation.

7 Experimental Evaluation

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

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

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
Classical ML						
nn-Meter [23]	<1% MAPE	1K/kernel	Hours/device	Device-specific	Medium	<1ms
XGBoost (TVM) [5]	20% MAPE	10K+	Online	Operator-level	Medium	<1ms
Deep Learning						
NeuSight [13]	2.3% MAPE	100K+	Pre-trained	Cross-GPU	Low	<10ms
Habitat [22]	11.8% MAPE	Online profiling runs	None (requires GPU)	Cross-GPU	Medium	Per-kernel profiling
Graph Neural Networks						
GRANITE [18]	0.97 τ	10K+	Hours	Cross-μarch	Low	<10ms
HELP [12]	1.9% MAPE	Meta-training	10 samples	Cross-platform	Low	<10ms
Transfer Learning						
LitePred [7]	0.7% MAPE	85 platforms	100 samples	85+ devices	Low	<1ms
Hybrid Analytical+ML						
Timeloop [17]	5–10%	Arch spec	None	Any accelerator	High	μs
ArchGym [10]	0.61% RMSE	Simulation	Surrogate training	Architecture-specific	Medium	ms
VIDUR [1]	<5%	Kernel profiles	Per-model	LLM-specific	High	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, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2024. Latency Predictors for Neural Architecture Search. In *Proceedings of Machine Learning and Systems (MLSys)*. 1–14.

[7] Yang Feng, Zhehao Li, Jiacheng Yang, and Yunxin Liu. 2024. LitePred: Transferable and Scalable Latency Prediction for Hardware-Aware Neural Architecture

Search. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 1–18.

[8] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*. 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[9] Jungho Kim et al. 2025. PyTorchSim: A Comprehensive, Fast, and Accurate NPU Simulation Framework. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14. https://doi.org/10.1145/3725843.3756045 PyTorch 2-integrated NPU simulator with custom RISC-V ISA and Tile-Level Simulation.

[10] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Norman P. Jouppi, Jignesh Parmar, Hyoukjun Kim, James Laudon, and Chandrakant Narayanaswami. 2023. ArchGym: An Open-Source Gymnasium for Machine Learning Assisted Architecture Design. In *Proceedings of the 50th International Symposium on Computer Architecture (ISCA)*. 1–16. https://doi.org/10.1145/3579371.3589049

[11] Hyoukjun Kwon, Prasanth Chatarasi, Michael Sarber, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2019. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. In *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–14. https://doi.org/10.1145/3352460.3358292

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] Ondrej Sykora, Alexis Rucker, Charith Mendis, Rajkishore Barik, Phitchaya Mangpo Phothilimthana, and Saman Amarasinghe. 2022. GRANITE: A Graph Neural Network Model for Basic Block Throughput Estimation. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. 1–13. https://doi.org/10.1109/IISWC55918.2022.00014

[19] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (2009), 65–76. https://doi.org/10.1145/1498765.1498785

[20] William Won, Taekyung Heo, Saeed Rashidi, Saeed Talati, Srinivas Srinivasan, and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-Model Training at Scale. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 283–294. https://doi.org/10.1109/ISPASS57527.2023.00035

[21] Yannan Nellie Wu, Joel Emer, and Vivienne Sze. 2022. Sparseloop: An Analytical Approach to Sparse Tensor Accelerator Modeling. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–15. https://doi.org/10.1109/MICRO56248.2022.00078

[22] Geoffrey X. Yu, Yubo Gao, Pavel Golber, and Asaf Cidon. 2021. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 503–521.

[23] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*. 81–93. https://doi.org/10.1145/3458864.3467882 Best Paper Award.

[24] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 863–879.