

Machine Learning-Driven Early Performance Prediction Framework for Accelerated Microarchitecture Simulation

| | | | | |
|--|---|---|--|--|
| Aiden Stickney <i>Texas A&M University</i> College Station, TX stickneyaiden@tamu.edu | Oswaldo Castro <i>Texas A&M University</i> College Station, TX ofcastro@tamu.edu | Aaron Chan <i>Intel Corporation</i> Austin, TX aaron.chan@intel.com | Paul V. Gratz <i>Texas A&M University</i> College Station, TX pgratz@tamu.edu | Jiang Hu <i>Texas A&M University</i> College Station, TX jianghu@tamu.edu |
| Aakash Tyagi <i>Texas A&M University</i> College Station, TX tyagi@tamu.edu | Jered Dominguez-Trujillo <i>Los Alamos National Laboratory</i> Los Alamos, NM jereddt@lanl.gov | Galen Shipman <i>Los Alamos National Laboratory</i> Los Alamos, NM gshipman@lanl.gov | Kevin Sheridan <i>Los Alamos National Laboratory</i> Los Alamos, NM kss@lanl.gov | |

Abstract—Rapid and accurate performance estimation is critical in evaluating novel microarchitectures, as it enables efficient exploration of architectural trade-offs. Unfortunately, traditional simulation techniques, while precise in predicting performance and power, incur tremendous slowdowns versus real machines. Despite prior works having explored machine learning-based performance prediction, the area remains far from sufficiently studied with existing approaches typically requiring large comprehensive datasets, frequent retraining, and heavy memory footprints with limited accuracy. Here, we introduce a new, fast and accurate, early-stage preview framework that uses partial simulation data, and leverages a smaller, faster tree-based machine learning (ML) model to forecast performance metrics such as IPC and Power. By training on a diverse set of configurations, our framework dynamically captures relationships between microarchitectural parameters in large OoO cores versus overall performance and other metrics. Collecting data from as few as 10 sample points taken during warmup, representing only 25 million instructions, our models achieve mean absolute percentage errors of 3-4%, preserving a majority of the model’s predictive accuracy while achieving a 25× speedup (96% reduction in simulation time). By comparison, linear regression techniques from the same point in simulation show an error of 50%. In cache DSE, we improve ranking accuracy by 25× compared to state-of-the-art prediction methods. Our results also show the proposed framework can accurately predict the performance of unseen (untrained) microarchitectural components including new prefetchers and branch predictors.

Index Terms—Computer architecture simulation, machine learning

I. INTRODUCTION

Modern processor design is increasingly difficult due to growing complexity, rising performance demands, and tightening power constraints. To capture a comprehensive measure of an architecture’s performance, it is advantageous to simulate as many instructions/cycles as possible. This is to overcome the initial fluctuations in components, such as branch predictors and prefetchers, that dynamically learn instruction patterns to optimize performance. Due to time-consuming simulations and

the iterative nature of the microarchitectural design workflow, architects require fast and cycle-accurate simulators to model performance and power over various designs, each with incremental changes. While seen as the standard for design-space exploration workflows, full-system simulators can take days to weeks to complete benchmarks, making performance and power modeling a significant bottleneck.

Prior work has pursued many strategies to accelerate microarchitectural simulation, from statistical sampling (SMARTS [29], SimPoint [21]) to regression models [9], [10], [25] and deep learning [7], [11], [22], [26]. More recent work has applied recurrent and graph neural networks for throughput estimation [16], [24], and ML frameworks for GPU performance prediction [1], [2], [18], [19], [28]. Transfer learning has been used to span architectures and accelerators [8], [15], [20], though often with heavy training and fine-tuning costs. Frameworks such as PerfVec [12] and SimNet [13] improve accuracy via instruction-level latency prediction but still require large datasets and complex models. Most recently, Concorde [17] proposed a compositional analytical-ML fusion approach for CPU performance modeling. Although many works claim to target design-space exploration, they often emphasize generalizing across workloads rather than across microarchitectural variants; a counterintuitive focus when the goal is to explore new designs. In contrast, our framework provides a lightweight, simulator-integrated approach that exploits warmup-phase sampling to reduce overhead while maintaining accuracy.

We propose a new method to accelerate performance and power modeling simulations by utilizing early-stage performance counters as input features to decision tree-based ML models. We show that while achieving a 25× speedup in simulation time, our framework maintains an average 3% error in cumulative instructions per cycle (IPC) and 12-14% error for power. While traditional methods forgo all warmup simulation-based features, these methods require extensive, manual, mi-

croarchitectural fine-tuning [12] to reach comparable, although worse error (4-7% IPC MAPE). In contrast, we show that, even from a brief preview during the warmup phase, machine learning models can accurately predict long-run performance and power metrics, reducing reliance on full-length simulations.

The novelty of our approach lies in using early-stage performance counters (the warmup preview) as input features to ML models. Although these counters are noisy while dynamic microarchitectural components train, the workload’s control-flow and memory-access tendencies are already representative enough for lightweight models to infer end-of-simpoint performance and power.

In cache design-space exploration, state-of-the-art methods that require fine-tuning [12] demand approximately 222 hours of data collection and training and exhibit a 3.6% misprediction rate. In contrast, our preview-based technique (Case Study 1, Section V-A) requires only 115 hours of data collection with no training overhead and reduces misprediction to 0.14%. This nearly halves exploration cost while improving ranking accuracy by 25×, underscoring the practicality of warmup-preview-based evaluation.

To make our approach immediately applicable, the preview window is integrated directly into the simulator as described in Section II. In our expected use case, the simulator ships with lightweight per-benchmark models, enabling users to run a brief warmup preview and obtain early performance and power estimates without retraining. Similar to how many trace-driven simulators provide benchmark traces, these models would be distributed by the simulator developers for community use. Given that benchmark suites such as SPEC [4], [23], and GAP [3] are released only once every several years, providing a single pre-trained model per benchmark is a relatively low, one-time overhead. Users wishing to evaluate entirely new benchmarks can still apply our methodology, incurring a one-time setup cost that is amortized over the long lifetime of the benchmark. To support reproducibility, our implementation is available at <https://github.com/AidenStickney/EarlyPerf>.

The paper makes the following contributions:

- First framework to accelerate simulation using warmup-preview sampling for ML inference across diverse microarchitectural design spaces.
- Extra Trees achieves best accuracy (3.22% MAPE vs. 9.47% last-value projection), within traditional simulation error bounds.
- Achieves 25× speedup with only 4% of simulation time and extends to MPKI, cache miss ratios, and dynamic power.

The paper is organized as follows: Section II reviews warmup behavior and its impact on early-stage performance. Section III introduces our framework and its implementation within ChampSim. Section IV presents the evaluation methodology and results. Section V demonstrates two case studies—cache design-space exploration and generalization to unseen microarchitectural components. Section VI concludes with a summary of contributions.

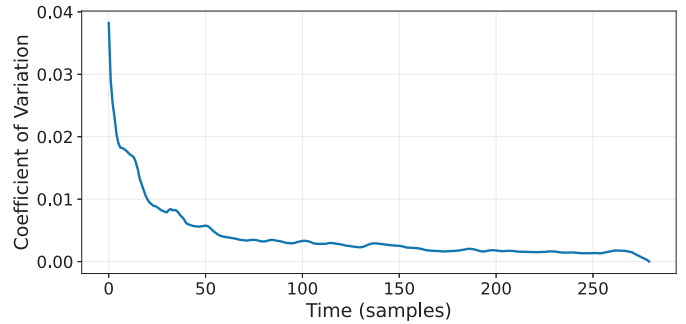


Fig. 1. Average sliding-window coefficient of variation of cumulative instructions per cycle (10-sample windows) across all 72 benchmarks.

II. BACKGROUND

Performance metrics such as IPC inherently exhibit a high level of variability from the initial training and convergence of dynamic microarchitectural components. To avoid such noise, it is common practice to ignore the early phase of simulation (the warmup phase). Using a sliding window of ten samples to present the coefficient of variation over time, Fig. 1 shows evidence of such variation in early metrics. The ‘end of warmup’, which is observed around the 75th sample, represents 187 million instructions, making up a considerable portion of the simulation. For this reason, architects require performance metrics from late-stage simulation, which can prolong runtimes.

III. PROPOSED FRAMEWORK

A. Overview of the Proposed Framework

To enable rapid and accurate performance and power prediction, our proposed framework is designed to modify the existing ChampSim simulator [6] by shipping with lightweight and pre-trained ML models. Models are trained on a per-workload basis, trading off workload generality for enhanced microarchitectural accuracy. That said, we note that the models are fully general with respect to the microarchitecture being simulated. In our proposed use case, the simulator is executed for a short segment from the beginning of the program or simpoint, which we dub the “warmup preview” (no further warmup needed). Early-stage performance counters taken from this segment of simulation and the microarchitecture configuration of the simulator are given to the ML model.

B. Feature Collection and Preprocessing

The framework utilizes early-stage performance metrics with machine learning inference, allowing metrics to be initialized at a baseline value and refined based on other insights and architectural choices. The ‘warmup preview’ window length introduces a tradeoff between the additional overhead of gathering metrics for a wide simulation window versus using a smaller simulation period to save time, at the cost of diminishing accuracy. Therefore, finding an optimal preview window is critical to balancing efficiency with precision, as further explored in Section IV-B. By default, for a single-core processor, ChampSim provides metrics for three key architectural components: the main core, last-level cache, and

DRAM. ChampSim outputs a set of counters, which we derive into model features as follows:

- **Main Core:** IPC (cumulative and instantaneous), branch misprediction rate (MPKI), predictor entries, L1D/L1I/ITLB/DTLB accesses and misses, 5-sample SMA of the target metric.
- **LLC:** Accesses and misses for reads/writes.
- **DRAM:** Memory-controller accesses, reads, and writes.

For model training, data sets are collected by running random microarchitecture configurations on a particular workload. ChampSim is modified to regularly sample performance counters for which the framework later parses with automated scripts. The additional simple moving average (SMA) of the target metric over a 5-sample preview window is included to better capture early-stage trends. The resulting framework is generally applicable to other microarchitecture simulators, not just ChampSim, provided similar hardware counters can be sampled.

To capture a thorough depiction of the vast architectural design space, we targeted a majority of the configurable parameters within ChampSim. For training, the chosen parameters and their value ranges are summarized in Table I. The resulting design space spans approximately 8×10^{39} unique architectural configurations; orders of magnitude larger than those explored in previous ML-based prediction frameworks. Ranges were selected based on realistic modern processor designs with additional overhead for the ever-evolving field and expanding performance demands.

Data collection requires balancing diversity and realism. While broader datasets improve generality, limiting to realistic configurations can yield greater accuracy within a fixed dataset size. Thus, some parameters were drawn from restricted distributions. Pipeline widths (fetch, decode, dispatch, execute, load/store, retire) were generated by selecting a random fetch width and adjusting subsequent stages with a Gaussian offset. L1D/I and LLC latencies were modeled using CACTI [27] based on cache size and associativity, targeting model accuracy for realistic use cases.

For components which require policy-based implementations such as branch predictors and prefetchers, the non-numerical component is represented as a nominal categorical feature, labeled with an integer representing the implementation. Although this approach treats the policy as a ‘black box’ without any characteristic policy features, insight is instead gathered from the categorical choice and early performance metric features. By abstracting the policy and instead relying on metrics, new unseen policies can be introduced without any fine-tuning while maintaining a high level of predictive accuracy.

C. Power Estimation

Along with performance metric estimation, we also explored using our technique to estimate full benchmark power consumption. In our test implementation, we leverage McPAT [14] as the baseline power model for us to estimate. The inputs to McPAT include architectural parameters (cache sizes, pipeline widths, branch predictor configurations), performance metrics

TABLE I
RANDOMIZED PARAMETERS AND RANGES DURING DATA COLLECTION

| Group / Parameter | Values / Ranges |
|-------------------------------------|--|
| Core Frequency (MHz) | 1000–5500 |
| Frontend Queues (Independent) | Fetch/Decode Buffers: 8–256; Dispatch Buffer: 2–256 |
| Widths (Independent) | Fetch/Decode/Dispatch/Exec/ LoadQ/StoreQ/Retire: 2–12 |
| Backend Resources (Independent) | Register File: 128–1024; ROB: 16–256, 320–1024; LSQ: 2–256; Scheduler: 2–512 (various) |
| Branch Predictor | Bimodal, Gshare, Hashed Perceptron, Perceptron, TAGE-SC-L |
| BTB | Return Stack Max: 48–80; Call Size Trackers: 768–1280; Indirect Pred Size: 3072–5120; Direct Pred Sets: 512–2048 |
| Dispatch Instr Buffer (Independent) | Window: 2–64; Sets: 2–64; Assoc: 2–16 |
| L1 (Independent) | Sets: 16–64; Assoc: 2–32; RW/Prefetch Q + MSHR: 2–128 |
| L1 Prefetchers | L1I: Next-line; L1D: Next-line, IP-stride, SPP, AMPM |
| LLC (Independent) | Sets: 512–8192; Assoc: 8, 16, 32; Prefetchers: Next-line, IP-stride, SPP, AMPM |
| Memory System (Independent) | Channels: 1, 2, 4, 8; Ranks: 1, 2, 4; DRAM Preset: DDR3/DDR4/DDR5 (126 total) with capacity 1-32 Gb, bus x4/x8/x16, freq 800–6400 MT/s |

(access counts, hits/misses, and utilization), and process technology. The input configuration and performance counters from ChampSim are written to McPAT for the final sampling window at 700 million instructions. The results from McPAT (e.g., dynamic power, leakage) are treated as the ground truth. A general-purpose, high-performance, technology configuration is used for McPAT.

Due to the non-linearity of power consumption, the raw power measurements follow skewed distributions, making prediction difficult for any ML model. Prior to model training, each target power metric (P) is transformed via a logarithm, $P_{ln} = \ln(P + 1)$. Log-transforming these values normalizes the distribution of ground truth metrics, improving predictive performance. After inference, the predicted value is converted back to the original scale by, $P = \exp(P_{ln}) - 1$.

D. Machine Learning Model Design

All models evaluated utilize two feature groups: the architectural configuration (see Table I) and performance metrics. Architectural components such as branch predictors and prefetchers, which are nominal, are categorically labeled with integers as discussed in Section III-B. Performance features are included as a vector from start of the simulation to the end of preview, containing samples every 2.5 million instructions.

TABLE II
ACCURACY OF CUMULATIVE IPC PREDICTION FOR VARIOUS ML MODELS
AVERAGED ACROSS ALL BENCHMARKS

| Model | MAPE (%) | R ² (%) |
|-----------------------|----------|--------------------|
| Last-Value Projection | 9.47 | 64.84 |
| Linear Regression | 49.79 | 4932.67 |
| Ridge Regression | 45.49 | 1768.43 |
| Trend Extrapolation | 240.98 | 28676.53 |
| Extra Trees | 3.22 | 97.23 |
| Random Forest | 3.40 | 97.10 |
| XGBoost | 4.11 | 96.20 |
| MLP | 31.24 | 260.62 |
| Transformer | 5.10 | 93.48 |

IV. EVALUATION

We train separate models for each of the 10 regression targets: cumulative IPC, mispredictions per kilo-instructions (MPKI), L1I read miss ratio, L1D read miss ratio, peak power, total leakage power, peak dynamic power, subthreshold leakage, gate leakage, and runtime dynamic power. This results in 10 pre-trained models for each of the 72 benchmarks. Trained models are evaluated using mean absolute percentage error (MAPE) against the full 700M-instruction ground truth for each regression target and the coefficient of determination (R²) is used as a secondary evaluation metric.

As previously discussed, evaluation is conducted on non-overlapping train/test splits utilizing cross-validation. The evaluation is organized as follows; we begin with an overview of the experimental setup. Next, an assessment of predictive techniques is conducted, comparing ML models to baseline methods for accuracy. We then determine an optimal preview duration and location, examining whether the additional simulation time for post-warmup previews is overshadowed by increased accuracy. The optimal model and preview duration are then evaluated per metric as well as for how the size of the dataset affects accuracy. Finally, the benefits of preprocessing power metrics through log normalization are evaluated.

A. Experimental Setup

We evaluate all supported SimPoints from the SPEC2006, SPEC2017, and GAP suites. Each benchmark runs for 700M instructions with 280 samples (every 2.5M instructions). Models are trained per workload so that all benchmarks contribute equally to training and testing. Training is performed on dual AMD EPYC 7443 nodes (4 cores/job), using two-fold cross-validation for XGBoost [5] and three-fold for others. Cross-validation ensures robustness by keeping training and testing strictly separate.

Baselines include last-value projection (final preview value vs. full simulation IPC), linear and ridge regression, and trend extrapolation, all trained to fit the preview window. For ML models, we test decision tree methods (Random Forest, XGBoost, Extra Trees), which train quickly and have small footprints, as well as two advanced models: a multi-layer perceptron (two hidden layers, 2.8k parameters) and a transformer (two encoder blocks, four attention heads, 64-dim embeddings, 15k parameters).

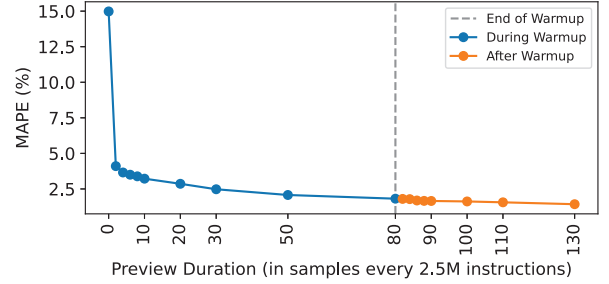


Fig. 2. Accuracy of cumulative IPC prediction for various preview durations using Extra Trees averaged across all 72 benchmarks.

TABLE III
ACCURACY OF EXTRA TREES MODEL FOR PREDICTING VARIOUS
PERFORMANCE METRICS AVERAGED ACROSS ALL BENCHMARKS

| Metric | MAPE (%) | R ² (%) |
|--|----------|--------------------|
| Cumulative IPC | 3.22 | 97.23 |
| Mispredictions per Thousand Instructions | 0.35 | 99.99 |
| L1D Read Miss Ratio | 0.67 | 94.29 |
| L1I Read Miss Ratio | 0.67 | 79.97 |

B. Model Selection and Preview Length Analysis

Table II summarizes the evaluated prediction techniques using a 10-sample preview window (25 million instructions) taken during warmup. Accuracy (MAPE) is averaged across 72 benchmarks. Baselines such as Last Value Projection and Trend Extrapolation, along with linear methods (Linear and Ridge regression), establish a lower bound of performance. Decision tree models, particularly an Extra Trees ensemble with 50 trees and a maximum depth of 10, consistently outperform all alternatives across the three benchmark suites while remaining efficient, requiring only 0.15 seconds to train and 13 milliseconds for inference.

Extra Trees and Random Forest excel because they capture non-linear patterns from early samples and smooth out warmup noise from caches and branch predictors, yielding accurate and stable IPC predictions. In contrast, linear regressions fail to capture phase behaviors, while Trend Extrapolation amplifies sampling noise. Deep learning (DL) models such as MLPs and Transformers can model complex non-linear data, but they require significant hyperparameter tuning and incur higher training and inference times. Despite this overhead, their accuracy improvements over tree methods are negligible. Overall, decision tree approaches offer the best trade-off between accuracy, robustness, and computational cost.

To evaluate preview duration, we varied the metric vector length from 0 to 50 samples and compared capturing metrics from warmup (sample 0) against post-warmup metrics (sample 80). The results are presented in Fig. 2. Although, post-warmup inputs yielded higher accuracy, the additional simulation time cancels out these gains, undermining the framework’s purpose. Within warmup, a 10-sample preview provides the best balance with a 25× speedup and 3.2% MAPE. This warmup configuration is used in subsequent evaluations.

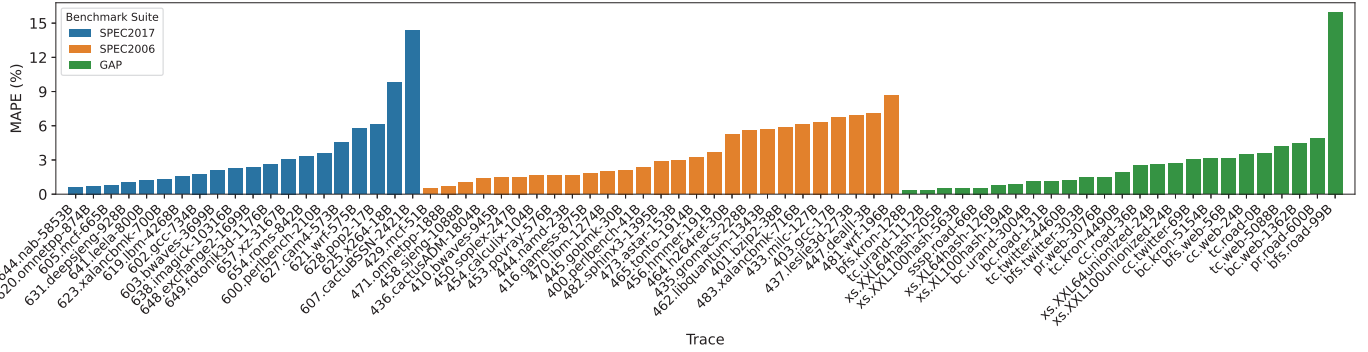


Fig. 3. Accuracy of cumulative IPC prediction using Extra Trees for all 72 benchmarks tested.

C. In-Depth Model Performance

The accuracy of the Extra Trees model with a 10 sample preview for various metrics is presented in Table III. The results show the model’s high accuracy for all predicted metrics, especially MPKI and the read miss ratios. Table IV displays the predictive accuracy of the model for power metrics. Although the power metrics exhibit a higher margin of error, the results are similarly promising and well within the bounds typically acceptable for early-stage power estimation. The per-benchmark averaged accuracy is included in Fig. 3.

To quantify the variability in prediction error, we also compute the standard error of the cumulative IPC MAPE across all benchmarks (treating each benchmark’s average MAPE as one sample). Despite the presence of outliers, the resulting standard error is approximately 0.06%, underscoring the model’s consistency in error across diverse workloads.

We also evaluated how training-set size influences accuracy. Results showed rapid improvement with additional configurations before reaching diminishing returns. About 300 configurations were sufficient, covering only $3.75 \times 10^{-36}\%$ of the full design space while still delivering strong predictive accuracy.

For power metrics, using McPAT [14] with log normalization significantly boosted prediction quality, improving accuracy by up to $60\times$ for peak dynamic power (see Table IV). While errors remain higher than IPC, the predictions reliably capture power-consumption trends across configurations, enabling practical early-stage design guidance.

D. Comparison with State of The Art

As shown in Case Study 1 (Section V-A), our framework consistently identified near-optimal cache configurations with higher accuracy and lower cost than PerfVec, providing the basis for comparison with state-of-the-art methods.

In recent years, ML-based performance modeling for microarchitectural acceleration has seen substantial progress. Whereas prior studies have largely focused on cross-workload generality, our framework demonstrates accuracy across microarchitectural variants, a focus more consistent with design-space exploration. Beginning with analytical models and regression techniques, the field has matured into widely adopting ML for simulation acceleration. Earlier approaches required

TABLE IV
MEAN ABSOLUTE PERCENTAGE ERROR (MAPE) FOR VARIOUS POWER METRICS WITH AND WITHOUT LOG NORMALIZATION

| Metric | MAPE With No Log Normalization (%) | MAPE With Log Normalization (%) |
|-----------------------|------------------------------------|---------------------------------|
| Runtime Dynamic Power | 293.18 | 14.35 |
| Peak Power | 736.50 | 11.98 |
| Peak Dynamic Power | 733.49 | 12.21 |
| Total Leakage | 8.29 | 3.55 |
| Subthreshold Leakage | 8.13 | 3.51 |
| Gate Leakage | 0.12 | 0.12 |

retraining to handle different architectures but generally generalized across workloads within a fixed design. The PerfVec [12] framework demonstrates promising accuracy for large-scale benchmarks but incurs high upfront costs, requiring comprehensive datasets of both traces and architectural variations. For larger architectural changes, additional fine-tuning may be needed, further increasing overhead.

In contrast, our framework prioritizes deployment and ease of use. Lightweight, fast, decision tree-based models trained on early-stage simulation metrics achieve high accuracy for performance and power predictions using only a fraction of simulation time. Without the need for fine-tuning, and in conjunction with ChampSim, the framework attains microarchitecture generality without sacrificing predictive accuracy. Although our models are trained per benchmark, limiting workload generalization, this tradeoff provides practical benefits (see Section V), as models can be efficiently updated when new workloads emerge. The framework also remains applicable to significant innovations such as new prefetcher or branch predictor designs, as shown in Section V-B.

Ultimately, the framework delivers a deployable prediction method tailored for fast-paced microarchitectural research and early-stage DSE. In real-world settings, where turnaround time is critical, it provides rapid insights into performance and power without the extensive training, dataset requirements, or fine-tuning demanded by state-of-the-art deep learning approaches.

V. APPLICATION

To evaluate the framework, we present two case studies representative of early-stage design space exploration.

A. Case Study 1: L1 and L2 Cache Size DSE

The first case study outlines a cache design space search for L1D and L2 cache sizes with the following parameter ranges:

- **L1D Cache Size:** 4KB, 8KB, 16KB, 32KB, 64KB, 128KB
- **L2 Cache Size:** 256KB, 512KB, 1MB, 2MB, 4MB, 8MB

Caches use realistic access latencies from CACTI [27]. Following PerfVec [12], we predict execution time and IPC on the first 100M retired instructions for direct comparison and rank L1D/L2 sizes using the objective function, $((1000 + 10 \cdot L1D_{Size} + L2_{Size}) \cdot 100M \text{ instructions}) / (\text{IPC} \cdot \text{Core Frequency})$ which seeks to evaluate the cache area and workload execution time. The final execution times from ChampSim are treated as the ground truth for comparison. In the objective function, the multiplier of 10 for the L1D cache size reflects a larger SRAM cell area, while the constant 1000 accounts for all other on-chip components.

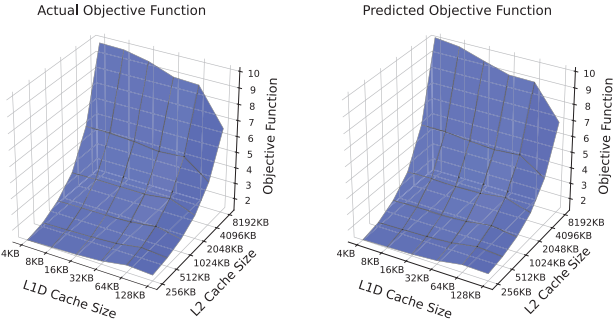


Fig. 4. Actual versus predicted objective function for L1D and L2 cache size sweep for the 444.namd-23B benchmark.

Evaluation of Case Study 1: The exploration was performed on 20 benchmarks (7 SPEC2017, 7 GAP, 6 SPEC2006). Our model ranked the true best configuration first in 19 benchmarks and within the top two for all 20. As shown in Fig. 4, predicted objective values closely match actual results.

PerfVec, by contrast, evaluated 17 benchmarks (3 reserved for fine-tuning) and selected the optimal configuration in only 4, ranked in the top two for 11, top three for 15, and top five for all, with 3.6% of designs outperforming its best choice. Our framework lowers this to 0.14%, yielding a 25 \times improvement in ranking accuracy.

Across the 20 benchmarks and 36 design points, there are $20 \times 36 = 720$ total configurations. With an average of 4 hours per simulation, the full exploration cost is $720 \times 4 = 3,600$ hours. Our framework instead simulates only the preview window (4% of each run), reducing the total cost to $0.04 \times 3600 = 115$ hours. By comparison, PerfVec must fine-tune on half the design space for three benchmarks before evaluating the remaining 17, requiring $3 \times 18 \times 4 = 216$ hours of simulation plus 6 hours for model training, totaling 222 hours. Overall, our

framework achieves nearly a 2 \times speedup in exploration time while also improving ranking precision, demonstrating clear practical benefits.

B. Case Study 2: Generalization to Unseen Architectural Components (Branch Predictors and Prefetchers)

Dynamic components in our framework are represented via early-stage performance metrics, allowing models to extract otherwise hidden information. Despite noise in these phases, decision tree-based models exploit these signals to infer DSE-relevant metrics without detailed knowledge of branch predictor or prefetcher implementations or additional fine-tuning.

This case study assesses the framework’s ability to generalize to unseen branch predictors and prefetchers. The case study is performed by removing all processor configurations with the selected state-of-the-art implementation (TAGE for branch prediction and SPP for prefetching) and training on the remaining configurations. The trained model is then tested using configurations with the unseen branch predictor or prefetcher. By testing our models on state-of-the-art implementations, we aim to depict a real development workflow for such components.

TABLE V
ACCURACY OF CUMULATIVE IPC PREDICTION USING EXTRA TREES FOR UNSEEN BRANCH PREDICTORS AND PREFETCHERS ON VARIOUS BENCHMARK SETS WITH A 10-SAMPLE PREVIEW DURING WARMUP

| Benchmark | Unseen Prefetchers | | Unseen Branch Predictors | |
|-----------|--------------------|--------------------|--------------------------|--------------------|
| | MAPE (%) | R ² (%) | MAPE (%) | R ² (%) |
| SPEC2017 | 2.75 | 98.21 | 4.41 | 79.03 |
| SPEC2006 | 3.14 | 97.20 | 5.74 | 79.53 |
| GAP | 2.59 | 97.49 | 4.19 | 80.81 |
| Average | 2.83 | 97.63 | 4.78 | 79.79 |

Evaluation of Case Study 2: Table V shows that the framework generalizes well to unseen microarchitectural choices, with slightly better accuracy for new prefetchers than for new branch predictors. Even in the harder branch-prediction case, sub-5% errors are sufficient to guide early-stage design decisions, confirming that our approach reliably infers cumulative IPC under both types of variation.

VI. CONCLUSION

We have demonstrated that using early-stage performance previews with tree-based ML models improves predictive accuracy compared to state-of-the-art methods. Furthermore, our results show that the proposed framework generalizes well across unseen microarchitectural variations. By shipping small, pre-trained models for each benchmark with the simulator, computer architecture design workflows will be accelerated by providing designers with rapid feedback on proposed designs.

ACKNOWLEDGMENTS

This work was partially supported by NSF (CCF-2106725, CCF-2425399, CCF-2529764) and Technical University of Munich Hans Fischer Senior Fellowship. Portions of this research were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing. Portions of this research were funded by a grant from Los Alamos National Laboratory.

REFERENCES

- [1] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 725–737. [Online]. Available: <https://doi.org/10.1145/2830772.2830780>
- [2] I. Baldini, S. J. Fink, and E. Altman, "Predicting gpu performance from cpu runs using machine learning," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, 2014, pp. 254–261.
- [3] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2017. [Online]. Available: <https://arxiv.org/abs/1508.03619>
- [4] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 41–42. [Online]. Available: <https://doi.org/10.1145/3185768.3185771>
- [5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. ACM, Aug. 2016, p. 785–794. [Online]. Available: <http://dx.doi.org/10.1145/2939672.2939785>
- [6] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," 2022. [Online]. Available: <https://arxiv.org/abs/2210.14324>
- [7] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XII. New York, NY, USA: Association for Computing Machinery, 2006, p. 195–206. [Online]. Available: <https://doi.org/10.1145/1168857.1168882>
- [8] A. Jones, A. Yazdanbakhsh, B. Akin, C. Angermueller, J. P. Laudon, K. Swersky, M. Hashemi, R. Narayanaswami, S. Chatterjee, and Y. Zhou, "Apollo: Transferable architecture exploration," in *ML for Systems Workshop at NeurIPS 2020*, 2020.
- [9] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "A predictive performance model for superscalar processors," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 39. USA: IEEE Computer Society, 2006, p. 161–170. [Online]. Available: <https://doi.org/10.1109/MICRO.2006.6>
- [10] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XII. New York, NY, USA: Association for Computing Machinery, 2006, p. 185–194. [Online]. Available: <https://doi.org/10.1145/1168857.1168881>
- [11] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPOPP '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 249–258. [Online]. Available: <https://doi.org/10.1145/1229428.1229479>
- [12] L. Li, T. Flynn, and A. Hoisie, "Learning generalizable program and architecture representations for performance modeling," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '24. IEEE Press, 2024. [Online]. Available: <https://doi.org/10.1109/SC41406.2024.00072>
- [13] L. Li, S. Pandey, T. Flynn, H. Liu, N. Wheeler, and A. Hoisie, "Simnet: Accurate and high-performance computer architecture simulation using deep learning," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 2, Jun. 2022. [Online]. Available: <https://doi.org/10.1145/3530891>
- [14] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: Association for Computing Machinery, 2009, p. 469–480. [Online]. Available: <https://doi.org/10.1145/1669112.1669172>
- [15] A. Marathe, R. Anirudh, N. Jain, A. Bhatle, J. Thiagarajan, B. Kailkhura, J.-S. Yeom, B. Rountree, and T. Gamblin, "Performance modeling under resource constraints using deep transfer learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: <https://doi.org/10.1145/3126908.3126969>
- [16] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithelmal: Accurate, portable and fast basic block throughput estimation using deep neural networks," 2019. [Online]. Available: <https://arxiv.org/abs/1808.07412>
- [17] A. Nasr-Esfahany, M. Alizadeh, V. Lee, H. Alam, B. W. Coon, D. Culler, V. Dadu, M. Dixon, H. M. Levy, S. Pandey, P. Ranganathan, and A. Yazdanbakhsh, "Concorde: Fast and accurate cpu performance modeling with compositional analytical-ml fusion," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1480–1494. [Online]. Available: <https://doi.org/10.1145/3695053.3731037>
- [18] K. O'neal, P. Brisk, A. Abousamra, Z. Waters, and E. Shriver, "Gpu performance estimation using software rasterization and machine learning," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, Sep. 2017. [Online]. Available: <https://doi.org/10.1145/3126557>
- [19] S. Pandey, L. Li, T. Flynn, A. Hoisie, and H. Liu, "Scalable deep learning-based microarchitecture simulation on gpus," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15.
- [20] S. Pandey, A. Yazdanbakhsh, and H. Liu, "Tao: Re-thinking dl-based microarchitecture simulation," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 2, May 2024. [Online]. Available: <https://doi.org/10.1145/3656012>
- [21] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using simpoint for accurate and efficient simulation," in *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 318–319. [Online]. Available: <https://doi.org/10.1145/781027.781076>
- [22] A. Renda, Y. Chen, C. Mendis, and M. Carbin, "Diffune: Optimizing cpu simulator parameters with learned differentiable surrogates," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 442–455.
- [23] C. D. Spradling, "Spec cpu2006 benchmark tools," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 130–134, 2007.
- [24] O. Sykora, P. M. Phothilimthana, C. Mendis, and A. Yazdanbakhsh, "Granite: A graph neural network model for basic block throughput estimation," 2022. [Online]. Available: <https://arxiv.org/abs/2210.03894>
- [25] M. Thazhuthaveetil, K. Vaswani, and P. Joseph, "Construction and use of linear regression models for processor performance analysis," in *Twelfth International Symposium on High-Performance Computer Architecture*. Los Alamitos, CA, USA: IEEE Computer Society, Feb. 2006, pp. 99–108. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/HPCA.2006.1598116>
- [26] Y. Wang, V. Lee, G.-Y. Wei, and D. Brooks, "Predicting new workload or cpu performance by analyzing public datasets," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 4, Jan. 2019. [Online]. Available: <https://doi.org/10.1145/3284127>
- [27] S. Wilton and N. Jouppi, "Cacti: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.
- [28] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "Gpgpu performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 564–576.
- [29] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: accelerating microarchitecture simulation via rigorous statistical sampling," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ser. ISCA '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 84–97. [Online]. Available: <https://doi.org/10.1145/859618.859629>