gem5 Co-Pilot: AI Assistant Agent for Architectural Design Space Exploration

Zuoming Fu*
Cornell University
Ithaca, NY, USA

Alex Manley*
University of Kansas
Lawrence, KS, USA

Mohammad Alian†
Cornell University
Ithaca, NY, USA

ABSTRACT

Generative AI is increasing the productivity of software and hardware development across many application domains. In this work, we utilize the power of Large Language Models (LLMs) to develop a co-pilot agent for assisting gem5 users with automating design space exploration. Computer architecture design space exploration is complex and time-consuming, given that numerous parameter settings and simulation statistics must be analyzed before improving the current design. The emergence of LLMs has significantly accelerated the analysis of long-text data as well as smart decision-making, two key functions in a successful design space exploration task. In this project, we first build gem5 Co-Pilot, an AI agent assistant for gem5, which comes with a webpage-GUI for smooth user interaction, agent automation, and result summarization. We also implemented a language for design space exploration, as well as a Design Space Database (DSDB). With DSDB, gem5 Co-Pilot effectively implements a Retrieval Augmented Generation system for gem5 design space exploration. We experiment on cost-constraint optimization with four cost ranges and compare our results with two baseline models. Results show that gem5 Co-Pilot can quickly identify optimal parameters for specific design constraints based on performance and cost, with limited user interaction.

KEYWORDS

Large Language Models, Design Space Exploration, Computer Architecture, gem5

1 INTRODUCTION

Computer architecture design space exploration (DSE) is the process of finding optimal parameters in a design space given a goal and constraints. In this project, we define a *design space* by its goals, parameters, and constraints, and DSE as the process (by human, machine, or both) of identifying the best parameters for a goal under constraints. Computer architecture DSE is simply this process instantiated in architecture design.

A typical example of DSE is exploring the cache hierarchy design space of a CPU microarchitecture to maximize performance under certain constraints (such as power and/or area) for a specific workload (for example, a matrix–matrix multiplication kernel).

Traditional human-driven DSE is tedious: architects must explore thousands of parameter values, run tens to hundreds of simulations, and analyze large volumes of statistics to understand the effects of different parameters. LLMs offer a way to accelerate this labor-intensive process. With their long-context capabilities, LLMs can process simulation data and scripts, provide well-reasoned decisions or suggestions, and, when paired with function-calling, act as autonomous agents that invoke external tools for more effective decision-making.

In this work, we present gem5 Co-Pilot, an LLM-powered assistant for computer architecture DSE using gem5 [4, 9]. To evaluate gem5 Co-Pilot, we explore the design space of an L2 cache under a total power constraint for a workload. Results show that gem5 Co-Pilot can reach near-optimal configurations within 1–8 generation stages (see Section 3.1.3) and 2–12 gem5 runs. Using GPT-4o via OpenAI's cloud API, each DSE session for a given constraint costs less than $0.5.

2 BACKGROUND

Computer architecture Design Space Exploration (DSE) aims to identify optimal design configurations under constraints such as performance, power, area, and cost. The vast and complex nature of the design space makes exhaustive evaluation impractical, driving the need for efficient and scalable exploration strategies.

Traditional DSE methods include brute-force search, parameter sweeps, and heuristic algorithms. Brute-force guarantees optimality but does not scale with problem size [19]. Heuristics such as simulated annealing and particle swarm optimization [5] focus search on promising regions, though they may converge to local optima and often require careful hyperparameter tuning. Grid search covers the parameter space systematically but suffers from exponential growth [7]. Architectural DSE often relies on slow cycle-accurate simulators like gem5, which further limits the practicality of these approaches. Domain-specific tools such as BOOM-Explorer [1] improve coverage but remain constrained by simulator overhead.

Machine learning offers more scalable alternatives by replacing simulations with predictive models. Techniques such as reinforcement learning (RL) [8], genetic algorithms [2], ant colony optimization [10], and Bayesian optimization (BO) [3] have been successfully applied. RL formulates DSE as a sequential decision process, adapting efficiently across workloads. Evolutionary and swarm intelligence methods enable efficient global exploration, while BO uses probabilistic surrogates for sample-efficient optimization. Frameworks like ArchGym [22] integrate multiple ML agents for trade-off optimization, and tools such as ZigZag [12] employ analytical modeling to avoid slow simulations for DNN accelerators. More recently, Pareto-driven active learning [21] combines active learning with Pareto optimality to accelerate multi-objective DSE.

Despite these advances, ML-based DSE still faces challenges, including high training cost, low volumes of training data, limited generalization, and interpretability issues. Real-world applications require not only efficiency but also explainability. The emergence of

*These authors contributed equally to this research.
†Corresponding author

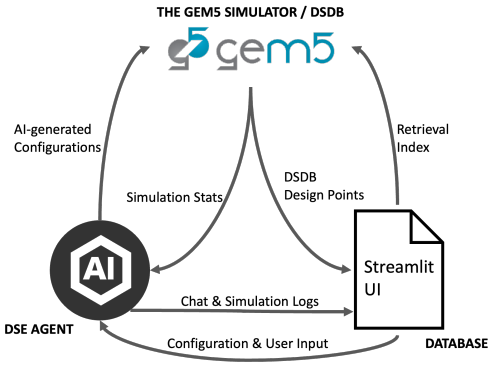


Figure 1: DSE system overview

generative large language models (LLMs) offers a promising direction for incorporating pre-trained reasoning into DSE workflows. Future efforts will likely focus on hybrid strategies (combining RL, BO, and heuristics) while incorporating additional constraints such as power, thermal, and security into the optimization process.

3 SYSTEM DESIGN

Our DSE system comprises three core components: a DSE AI agent, the gem5 simulator (or DSDB, see Section 3.3), and a Streamlit-based user interface (UI), as illustrated in Figure 1. The architecture separates the agent, simulation unit, and UI to improve flexibility and extensibility. The DSE agent, built around an LLM-driven state machine, dispatches gem5 configuration parameters to the simulator and relays contextual data (e.g., chat history) and simulation logs to the UI. The gem5 simulator (or DSDB) returns simulation results to the agent for optimization. The Streamlit UI retrieves design points from the DSDB for visualization and manages system configuration and user inputs for the agent.

As depicted in Figure 2, the AI agent acts as the central controller. A system prompt defines high-level goals, including DSE rules and permitted states, to steer the LLM's operation. This prompt, combined with chat history, provides context for each LLM invocation. A state machine with four states (`ANA`, `GEN`, `QA`, and `EXIT`), coupled with a prompt bank (see Section 3.1.3), enables the LLM to behave correspondingly across different functions , balancing flexibility with control.

The simulation backend evaluates system performance using gem5. The current incarnation of gem5 Co-Pilot supports gem5 simulations in System Call Emulation (SE) mode. Configuration parameters are converted into gem5 execution commands, and results, including performance statistics, power, and area, are returned to the agent. To minimize simulation overhead, the DSDB supplies precomputed results, significantly speeding up exploration.

A Streamlit-based web UI [16] supports system setup, simulation control, and visualization. Its interactive features (such as dynamic plotting and public URL access) facilitate remote use and enhance overall accessibility.

3.1 DSE AI agent

The AI agent powers the system, executes the explorations, and analyzes the history parameters and stats. It handles the most complex task of DSE and utilizes the full power of LLMs. However, the

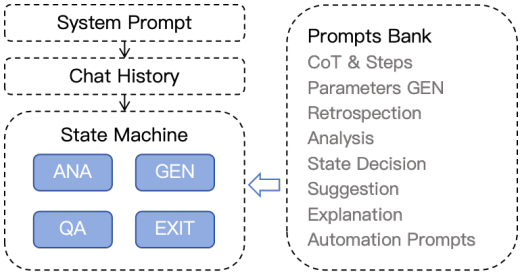


Figure 2: DSE agent Structure

LLM itself could hallucinate and generate inaccurate output. To solve these potential problems, we applied several techniques to our AI agent.

3.1.1 Automation mode switching. We provide a mode selection button that allows the user to choose between automation and manual modes at any time. In this way, the user can decide whether to let the agent run autonomously or to interact with it for a more customized DSE experience. By default, the agent operates in manual mode, under which an input box is displayed. Upon switching to automation mode, the input box is hidden, and the agent begins running automatically using preset prompts. If the user switches back to manual mode, the agent will wait for the current automation process to complete before displaying the input box again.

The automation prompts are listed in Table 1. It should be noted that the prompt in the ANA state is executed at the end of that state to transition to the GEN state, as is the case for other automation prompts. We use automation mode for all experiments to eliminate human influence on the results, while keeping manual mode available for interactive exploration.

Table 1: Prompts for automation mode

State	Prompts
`ANA`	Based on the analysis, generate a new parameters batch for the mission.
`GEN`	Analyze all the simulation results so far (if we have them) and provide insights that would help better design.
`QA`	I have no more questions. Please go to ANA state.
`EXIT`	Provide the final parameters batch (the best one) (of size 1) for the mission and exit the program.

3.1.2 Concurrent simulations. Multiple threads can be used to speed up DSE by running multiple gem5 commands concurrently. gem5 Co-Pilot is built to suit this feature. We design the agent to be able to generate multiple configurations and get their simulation results in each `GEN` state (see Section 3.1.3). We define `parameters_set` as a set of parameters that can be used for a simulation. We define `parameters_batch`, or `batch`, as all the `parameters_set` that the agent generated in a `GEN` state. The number of `parameters_set` in a `parameters_batch` is called the batch size. Accordingly, we define a `results_batch` as all the concurrent results that the simulator returns in a `GEN` state. gem5 Co-Pilot allows up to 20 concurrent simulations.

3.1.3 State machine. A state machine structures the LLM's task execution through a clear, state-driven process. Each state corresponds to one response and is bound to a structured output. The system prompt specifies the initial state, all possible states, and the transitions between them. The LLM's role in each state is provided in Table 2.

Table 2: Agent States and Their Responsibilities

State	Description
`ANA`	- Retrospect past simulation results from chat history. - Provide CoT-based step-wise analysis and conclusions. - Explain reasoning and suggest next state (`ANA` or `GEN`).
`GEN`	- Generate a `parameters_batch` based on analysis. - Each `parameters_set` respects design space ranges. - Batch size fits concurrent simulation limit. - Suggest next state (`ANA` or `GEN`).
`QA`	- Answer user's questions using history and known context. - Transition back to `ANA` after answering.
`EXIT`	- Output final best `parameters_batch` (of size 1). - Triggered when no further improvement is expected.

3.1.4 LLM structured output. A major challenge in using LLMs for parameter generation is their occasional failure to adhere to required formats or content (even with explicit prompting) as observed in earlier models like GPT-3. This is particularly problematic for DSE, which demands interpretable and highly stable parameter outputs. Additionally, the system must output the next state and the LLM's reasoning behind its parameter choices.

Modern LLMs such as GPT-4 and GPT-4o address this issue through structured output [14], which enables the model's response to be directly parsed into a Python dictionary. This approach not only ensures consistent and interpretable responses but also reduces output length, thereby lowering generation costs.

3.1.5 Results Retrospection. Results retrospection (RR) is a technique used in our LLM prompting: in every `ANA` state, instead of directly asking the agent to provide an analysis on history results, which were stored in history chat, we add a "retrospection" prompt:

> "You should first retrospect on all history simulation results. They are in the chat history, you should read them and print them in batches in a timely manner."

to let the LLM retrospect on historical results before providing the analysis to prevent LLM from forgetting historical information as the chat length gets longer.

3.1.6 Baseline Preservation. Baseline Preservation (BP) means that in every `GEN` state, instead of asking the LLM to directly generate a `batch` without any order, an additional prompt:

> "When the number of concurrent simulations is larger than 1, you should make the first parameters_set the best one you can figure out, and use other ones for exploration."

who asks the AI to preserve the possible best `parameters_set` in the first thread while keeping other threads for wild exploring is applied to prevent chaos in the DSE process.

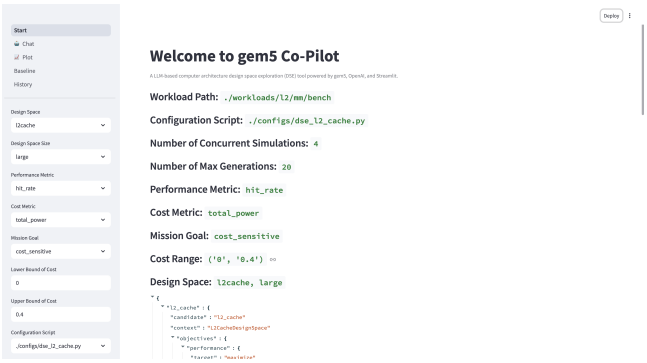


Figure 3: Streamlit UI

3.1.7 Chain of Thought Reasoning. A major challenge in employing LLMs for design space exploration (DSE) is ensuring logically sound and accurate outputs. Chain of Thought (CoT) prompting [18] addresses this by guiding the model to reason step-by-step, improving deliberation over one-shot responses.

In DSE, CoT enables the LLM to systematically evaluate tradeoffs. For example, it can reason that increasing L2 cache size typically raises area and power costs, and should be avoided if performance gains are marginal. Such structured reasoning is essential for making informed parameter selections within constrained design spaces.

3.1.8 Streamlit User-Interface. We use a popular framework used by today's web-based AI Applications, Streamlit [16], to build our UI, as shown in Figure 3. It is powerful for displaying chat-bots, DataFrames, dynamic 2D and 3D graphs, curve line analysis, and so on. It is suitable for our DSE setup, interaction, analysis, and result display.

3.2 The gem5 Simulator

The gem5 simulator is a modular, open-source framework widely used for computer architecture DSE. It provides a flexible tool for simulating CPUs, memory hierarchies, interconnects, and devices.

In our framework, gem5 executes binaries under various configurations generated by the AI agent, producing detailed runtime statistics and performance counters. These outputs are passed to McPAT [6] for joint power and area estimation, enabling comprehensive design point characterization within constrained optimization objectives.

3.3 DSDB: A Design Space Database for gem5

To keep track of the design space exploration, gem5 Co-Pilot introduces a Design Space Database (DSDB), which is a structured repository of comprehensive simulation results from previously executed runs. The first important use case of DSDB is to establish baseline Pareto-optimal frontiers for DSE. gem5 Co-Pilot also leverages the DSDB to estimate the performance of a future simulated system without executing a full simulation, significantly accelerating the DSE process.

In our evaluation, the DSDB provides precomputed Pareto frontiers for benchmarking, enabling objective assessment of gem5 Co-Pilot 's optimization performance. It also accelerates the gem5 Co-Pilot loop by retrieving prior results, substantially reducing

exploration time and promoting convergence. Although results are fetched from pre-run simulations, each entry corresponds to an actual simulation, ensuring rigor while maintaining efficiency. The simulations populating the DSDB were conducted on a SLURM-managed cluster, as specified in Table 3.

4 METHODOLOGY

Table 3: Simulation Environment Configuration

Field	Specification
CPU Model	Intel Xeon E5-2660 @ 2.20 GHz (per node)
Cores per Job	1 core per SLURM task
Memory per Job	4 GiB DRAM
Operating System	Rocky Linux 9.4
Execution Model	Cluster-parallel: one simulation per SLURM job

4.1 L2 Cache Design Space Specification

Table 4 defines the large L2 cache design space used in all subsequent experiments. This space spans five key microarchitectural parameters: cache size, associativity, number of MSHRs, MSHR target count, and replacement policy. Each parameter is assigned a broad and representative range of values, capturing realistic design options found in modern systems. When all combinations are enumerated, the design space contains over 7,770 unique configurations, enabling thorough exploration of architectural trade-offs across performance, power, and area.

Table 4: L2 Cache Design Space (Large)

Parameter	Values
`l2_size`	128KiB, 256KiB, 512KiB, 1MiB, 2MiB
`l2_assoc`	2, 4, 8, 16, 32
`l2_mshrs`	16, 32, 64
`l2_mshr_tgts`	6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32
`l2_policy`	LRURP, RandomRP, FIFO, FIFORP, LFURP, BIPRP, MRURP, BRRIPRP, SecondChanceRP

4.2 DSE Goal and Metrics

Architectural design is all about trade-offs. We set our system goal to a function of performance and cost `GOAL = f(PERF, COST)`. The performance and cost are two entangled variables which in a DSE problem architects must take into consideration. In our study, we set our `GOAL` to be finding the design point that achieves the highest performance in a given cost range. We call this point the performance-under-constraint point ($\mathtt{P_{Constraint}P}$) and find that it is both DSE-representative and LLM-friendly, due to the concept being easily understood by the model. There may be more than one $\mathtt{P_{Constraint}P}$ in a given cost range and we regard them all as valid and reasonable goals.

To evaluate the effectiveness of gem5 Co-Pilot, we use `l2_hit_rate` as the performance (`PERF`) metric. The `l2_hit_rate` is provided by gem5 stats and is calculated in Equation 1.

$$PERF_{\text{hit\_rate}} = 1 - R_{\text{miss}} \tag{1}$$

R_{miss} = `stats.l2cache.overallMissRate.total`

We choose `total_power` as our `COST`. It is provided by the McPAT power modeling tool and calculated in Equation 2.

$$COST_{\text{total\_power}} = P_{\text{dyn}} + P_{\text{gate}} + P_{\text{sub}} \tag{2}$$

P_{dyn} = `mcpat.total_l2s.runtime_dynamic`
P_{gate} = `mcpat.total_l2s.gate_leakage`
P_{sub} = `mcpat.total_l2s.subthreshold_leakage`

We use `(perf_ratio, nsims)` pairs as the metric for our goal. We define `perf_ratio` as the proportion of a DSE agent generated `PERF` result value to the `PERF` value of $\mathtt{P_{Constraint}P}$ in a given range of `COST`. To be more specific, when we do DSE within `COST` range [0,0.15] , if the DSE agent finally finds a potential best design point that achieves a `PERF` value of P_{DSE}, while the ground truth best `PERF` value of $\mathtt{P_{Constraint}P}$ is $P_{\mathtt{P_{Constraint}P}}$, then:

$$\mathtt{perf\_ratio}_{[0,0.15]} = \frac{P_{DSE}}{P_{\mathtt{P_{Constraint}P}}}$$

In our experiment, a `perf_ratio` higher than 97% is regarded as a realistic design point.

`nsims` is the number of simulations the agent applies before a DSE achieves its final result. The number of GEN stages used is also recorded as it measures the speed of convergence. The relationship between the number of GEN stages and `nsims` can be shown as:

$$\mathtt{nsims} = nGENs \times concurrent\_sims\_per\_GEN$$

Overall, the `(perf_ratio, nsims)` pairs measure a system's ability to achieve a good result while considering the simulation cost.

4.3 Experiment Design

We evaluate our approach using a series of cost ranges: `[0,0.12]`, `[0,0.15]`, `[0,0.2]`, `[0,0.4]`, which represent distinct regions of the cost-performance Pareto front. For each range, we conduct DSE using both baseline methods and gem5 Co-Pilot, recording the achieved `perf_ratio` and number of simulations (`nsims`).

We compare against two baseline methods: Random Search (RS) and a Genetic Algorithm (GEN) [11]. For RS, we report the first configuration achieving `perf_ratio` >97% and its simulation cost. The GEN implementation uses a population of 50 over 20 generations, with 80% crossover and 20% mutation rates, employing tournament selection and elitism. All candidates undergo dynamic validity checks against the $\mathtt{P_{Constraint}P}$ objective.

We further examine the impact of concurrent simulation count and key prompt strategies (Results Retrospection-RR and Baseline Preservation-BP). Each configuration was run three times, with the best result selected to account for LLM randomness.

A small number of runs failed due to rare LLM output irregularities (e.g., malformed Python dictionaries). These were automatically detected and corrected by requesting a renewed response, ensuring continuous operation.

All simulations use the gem5 System Call Emulator (SE) mode. The baseline configuration (Table 5) features an O3 CPU, two-level

caches, and DDR3 memory, with several L2 parameters available for DSE tuning.

The evaluation workload is a blocked matrix multiplication (128×128 matrices, block size 32) in C. With a footprint of ~393 KB (exceeding L2 capacity), it stresses the memory hierarchy while maintaining short runtimes.

Experiments were conducted on both a SLURM cluster and a high-end workstation (Intel i9-13900HX, 64GB RAM). The full design space (7,770 configurations) completes in ~6 hours on the cluster. On the workstation, a single simulation takes ~1 minute, and a full DSE (20 generations) finishes in ~30 minutes.

The OpenAI API cost for a full automated DSE run remains below $0.50 USD, compared to over $8 for even a brief manual DSE (10-minute) under a $50 per hour labor cost , demonstrating significant cost efficiency.

Table 5: Baseline gem5 System Configuration

Component	Configuration
ISA	X86
CPU model	O3 (out-of-order)
Clock frequency	1 GHz
Memory mode	Timing
Main memory size	512 MiB
L1 I-cache	16 KiB, 2-way, latency: 2 cycles
L1 D-cache	64 KiB, 2-way, latency: 2 cycles
L2 cache	256 KiB, 8-way, latency: 20 cycles (DSE tunable)
L2 MSHRs / Targets per MSHR	20 / 12 (DSE tuneable)
L2 Replacement policy	LRU (DSE tuneable)
System buses	L2XBar and SystemXBar
Memory type	DDR3_1600_8x8

5 RESULTS

5.1 Performance-Cost Optimization

Our experimental evaluation demonstrates the effectiveness of gem5 Co-Pilot in navigating the architectural design space compared to traditional approaches. The results highlight both the efficiency and optimization capabilities of gem5 Co-Pilot across different cost constraints.

Figure 4 presents the Pareto-optimal frontier for the full design space with Co-Pilot's selected design points overlaid at 3 and 4 concurrent simulations. These configurations demonstrate gem5 Co-Pilot's ability to effectively navigate the performance-cost trade-off space while utilizing different levels of parallel simulation resources. The selected points cluster near the Pareto frontier across all cost ranges, indicating effective optimization under varying constraint levels.

Note that we evaluated all possible design points to obtain a complete view of the design space and then applied the skyline algorithm to determine the ground-truth Pareto frontier. We subsequently used gem5 Co-Pilot to generate the optimal design point within a given cost range and observed that the generated points lie on the frontier. We perform this exhaustive evaluation in our paper to demonstrate that gem5 Co-Pilot can indeed identify optimal design points. In practice, however, users do not need to repeat this process, as our results have already validated gem5 Co-Pilot 's ability to discover frontier points effectively.

Table 6: Performance-Cost Optimization Results (Performance ratio and simulation utilization)

Model	Cost Range (W)			
	[0,0.12]	[0,0.15]	[0,0.2]	[0,0.4]
RS	0.978 (25)	0.986 (11)	0.980 (22)	0.997 (6)
GEN	1.000 (133)	1.000 (158)	1.000 (137)	1.000 (127)
C-P-1	0.980 (6×1)	0.987 (5×1)	0.989 (8×1)	0.999 (2×1)
C-P-2	0.978 (5×2)	0.992 (6×2)	0.989 (6×2)	0.998 (1×2)
C-P-3	0.974 (4×3)	0.987 (3×3)	0.980 (3×3)	1.000 (1×3)
C-P-4	0.987 (2×4)	0.986 (3×4)	0.973 (1×4)	1.000 (1×4)

Note: Values show `perf_ratio` achievement (higher is better) and required simulations ($nsims = nGENs \times concurrent\_sims\_per\_GEN$, in parentheses). RS = Random Search baseline; GEN = Genetic Algorithm baseline, C-P-i = Co-Pilot with i concurrent simulations (1-4). All Co-Pilot variants use the DSDB framework.

Table 6 compares the performance ratios and simulation counts across different methods and cost ranges. The Genetic Algorithm (GEN) achieves perfect optimization (1.0) but requires substantially more simulations (127–158). Random Search (RS) attains high performance (0.978–0.997) with moderate simulation counts (6–25), though with greater variability.

Co-Pilot demonstrates superior efficiency, achieving near-optimal performance (0.973–1.000) with significantly fewer simulations (as low as 1–8). Key observations include:

- **Rapid Convergence**: Co-Pilot reaches >97% of optimal within 1–8 simulations, vastly outperforming RS (11–25) and GEN (127–158). For example, in the tight [0,0.12] range, C-P-4 attains 98.7% with only 2 simulations.
- **Scaling with Concurrency**: Higher concurrency (e.g., C-P-3, C-P-4) proves that gem5 Co-Pilot performs well under different concurrences and total simulation count decreases consistently as concurrency increases, especially under cost range [0,0.12], [0,0.15], and [0,0.02].
- **Constraint Sensitivity**: All methods perform best in the widest cost range [0,0.4], where Co-Pilot achieves nearly perfect results in just 1–2 simulations.

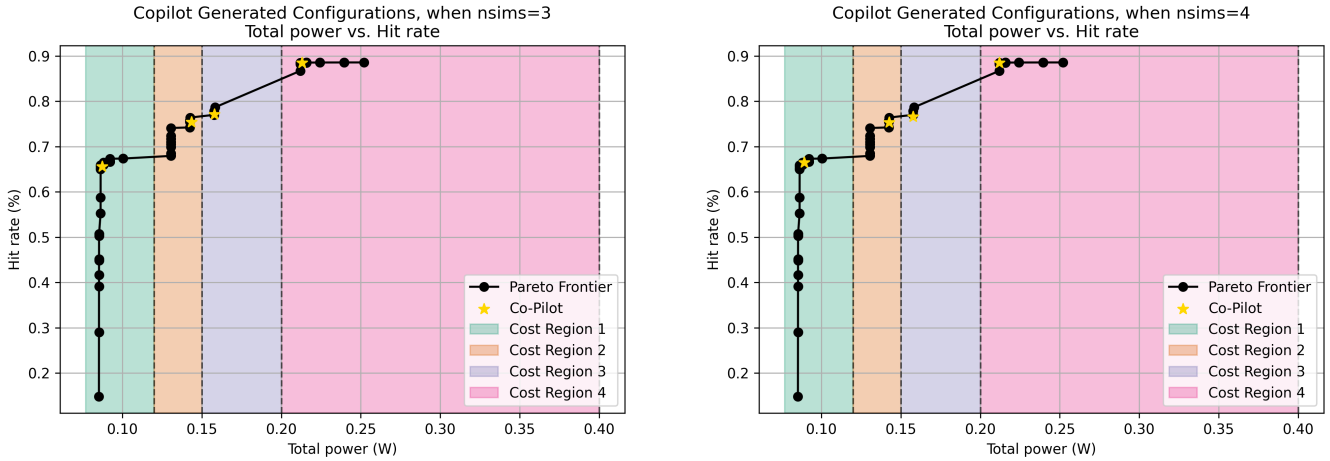
5.2 Ablation Study

We conduct an ablation study to quantify the contribution of individual components in gem5 Co-Pilot, focusing on key prompt engineering elements and their impact on design space exploration.

Two critical prompt components are evaluated through removal experiments, where each is systematically omitted and the resulting system is compared against the full configuration:

- **Results Retrospection (RR)**: Removes the result retrospection (see Section 3.1.5) in the `ANA` state. The agent does not review historical simulation results.
- **Baseline Preservation (BP)**: Removes baseline preservation (see Section 3.1.6) in the `GEN` state. The agent does not retain the best-performing `parameters_set` in the initial thread.

Table 7 summarizes the performance impact of ablating key prompt components across cost ranges. Results show that Results



(a) 3 Concurrent Sims **(b) 4 Concurrent Sims**

Figure 4: Co-Pilot generated points (yellow) when optimizing total power vs. hit rate, superimposed on the full Pareto frontier (black) under 3 and 4 concurrent simulations. The Pareto frontier is extracted from the DSDB dataset. The cost thresholds—[0, 0.12], [0, 0.15], [0, 0.2], and [0, 0.4]—represent different ascending stages along the frontier. The experiment system uses an O3 CPU with 2-level caches running a blocked matrix multiplication benchmark (see Section 4.3).

Table 7: Ablation Study Results (`perf_ratio` degradation) under Co-Pilot 3-sim

Ablation	Cost Range (mW)			
	[0,0.12]	[0,0.15]	[0,0.2]	[0,0.4]
Full System	0.974 (3)	0.987 (3)	0.980 (3)	1.000 (1)
No RR	0.000 (∞)	0.970 (1)	0.944 (4)	1.000 (1)
No BP	0.974 (6)	0.987 (8)	0.959 (8)	1.000 (2)
No RR+No BP	0.974 (13)	0.972 (3)	0.942 (5)	1.000 (1)

Note: Values represent (perf_ratio achieved / GEN count). Simulation per GEN is fixed at 3. RR = Results Retrospection (historical simulation analysis), BP = Baseline Preservation strategy. "∞" indicates failure to converge within 20 simulation cycles. All experiments use the same initial seed configuration. The [0,0.12] range shows highest sensitivity to component ablation.

Retrospection (RR) is critical under tight constraints: removing RR alone in the [0,0.12] mW range causes complete failure (`perf_ratio` = 0), while also degrading performance in [0,0.15] and [0,0.2] ranges. Baseline Preservation (BP) mainly improves efficiency—without BP, the number of generations increases by 100–167% across most ranges, though final `perf_ratio` remains high (≥97.4% in three ranges). Their interaction is complementary: BP partially compensates for RR's absence in strict constraints, maintaining performance at the cost of more simulations. In relaxed constraints ([0,0.4] mW), both components become unnecessary, as all configurations achieve `perf_ratio` = 1.0 within 1–2 simulations.

6 RELATED WORK

Recent studies have explored using large language models (LLMs) in hardware design and design space exploration (DSE). For high-level synthesis (HLS), LLM-DSE [17] proposed a multi-agent system that optimizes compiler directives through specialized roles. Although effective, such approaches remain at a high abstraction level and depend heavily on synthesis feedback from tools like Merlin and Vitis.

Other works, such as CodeChain [20], generate Verilog or HLS code from natural language prompts. These typically perform one-time synthesis with minimal iteration and often omit integration with performance or power evaluation.

In comparison, our work targets cycle-accurate architectural exploration. The gem5 Co-Pilot embeds LLM reasoning within a state-managed exploration loop that directly manipulates hardware parameters. Each design is assessed using gem5 for performance and McPAT for power and area, creating a closed-loop refinement process based on simulation feedback. Unlike scripted methods [15], Co-Pilot employs natural language planning to interpret trade-offs and navigate the design space efficiently.

This approach aligns with the agentic, cross-layer vision of Architecture 2.0 [13], where AI agents use system feedback to guide optimization. Co-Pilot implements this through declarative design specification, intelligent control, and cycle-level evaluation, bridging high-level intent and low-level implementation in architectural DSE.

7 CONCLUSION

In this work, we introduced gem5 Co-Pilot, a generative AI-powered assistant for architectural design space exploration using gem5. We evaluated gem5 Co-Pilot with a simple yet tractable example of design space exploration for efficient cache design and demonstrated its effectiveness. gem5 Co-Pilot can significantly reduce the time required to explore the design space of future hardware architectures by intelligently navigating different configurations—avoiding unnecessary simulations whose results can either be predicted (by maintaining a design space database) or ruled out through reasoning. The design of gem5 Co-Pilot is modular, making it both extensible and flexible for future use not only with gem5 but also with other architectural tools.

ACKNOWLEDGMENTS

This work was supported in part by NSF award 2519295. We thank Ampere Computing for their generous server donation that we used to run simulations. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2021. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework. In *2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[2] C. Basile, D. Bertozzi, A. Jantsch, and M. Sami. 2003. Automatic design space exploration using genetic algorithms. In *Design, Automation and Test in Europe Conference and Exhibition*. IEEE, 274–279.

[3] R. Benassi, J. Bect, and E. Vazquez. 2011. Efficient multi-objective design space exploration using surrogate models. *Structural and Multidisciplinary Optimization* 43, 4 (2011), 507–525.

[4] 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. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[5] David Brooks and Margaret Martonosi. 2001. Automated design space exploration using simple analytic models. In *Proceedings of the 2001 International Conference on Supercomputing (ICS)*. ACM, 24–33. https://doi.org/10.1145/377792.377801

[6] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 469–480.

[7] Xiaoqiao Li, Tulika Mitra, and Abhik Roychoudhury. 2004. Design Space Exploration of Caches Using Compressed Traces. In *Proc. 18th Int'l Conf. on Supercomputing (ICS)*. 243–252.

[8] Hanrui Liu, Haicheng Wei, Yujun Guo, Jiacheng Wu, and Vivienne Sze. 2021. Hardware architecture search with reinforcement learning. *International Conference on Learning Representations (ICLR)* (2021).

[9] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. 2020. The gem5 Simulator: Version 20.0+. arXiv:2007.03152 [cs.AR] https://arxiv.org/abs/2007.03152

[10] Pedro Marujo, Luiz Zschornack, Ney Calazans, and Fernando Moraes. 2011. An ant colony optimization approach to design space exploration of network-on-chip architectures. In *Proceedings of the 2011 International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 313–322.

[11] Hossein Mehri and Bijan Alizadeh. 2017. Genetic-Algorithm-Based FPGA Architectural Exploration Using Analytical Models. *ACM Transactions on Design Automation of Electronic Systems* 22, 1 (2017), 1–17. https://doi.org/10.1145/2939372 Published online September 2 2016.

[12] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. 2021. ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators. *IEEE Trans. Comput.* 70, 8 (2021), 1160–1174. https://doi.org/10.1109/TC.2021.3059962

[13] Vijay Janapa Reddi and Amir Yazdanbakhsh. 2025. Architecture 2.0: Foundations of Artificial Intelligence Agents for Modern Computer System Design . *Computer* 58, 02 (Feb. 2025), 116–124. https://doi.org/10.1109/MC.2024.3521641

[14] Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2997, 13 pages.

[15] Renan Tashiro and Marcio Seiji Oyamada. 2016. An Environment for Design Space Exploration Using gem5-McPAT. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. 220–225. https://doi.org/10.1109/SBESC.2016.042

[16] Adrien Treuille, Amanda Kelly, Thiago Teixeira, et al. 2019. Streamlit: The fastest way to build data apps. https://streamlit.io. Accessed: 2025-06-28.

[17] Hanyu Wang, Xinrui Wu, Zijian Ding, Su Zheng, Chengyue Wang, Tony Nowatzki, Yizhou Sun, and Jason Cong. 2025. LLM-DSE: Searching Accelerator Parameters with LLM Agents. arXiv:2505.12188 [cs.AR] https://arxiv.org/abs/2505.12188

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[19] Marilyn Wolf, Ahmed A Jerraya, and Grant Martin. 2005. A survey of design space exploration for hardware systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 10, 3 (2005), 273–279.

[20] J.D. Zamfirescu-Pereira, Eunice Jun, Michael Terry, Qian Yang, and Bjoern Hartmann. 2025. Beyond Code Generation: LLM-supported Exploration of the Program Design Space. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. ACM, 1–17. https://doi.org/10.1145/3706598.3714154

[21] Jianwang Zhai and Yici Cai. 2023. Microarchitecture Design Space Exploration via Pareto-Driven Active Learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023). https://doi.org/10.1109/TVLSI.2023.3311620 Early Access / preprint in 2023.

[22] Xin Zhao, Arka Mishra, Aditya Bagaria, Amirali Ebrahimi, Udit Thakker, Hanrui Zhang, Yuke Zhang, Jason Mars, Krste Asanović, Vivienne Sze, et al. 2022. ArchGym: An Open-Source Gym for Machine Learning-Driven Architecture Research. *arXiv preprint arXiv:2205.10371* (2022).