Agentic LLMs for Microarchitecture Research: The Role of the Human Architect

Georgios Vavouliotis
Computing Systems Lab, Huawei Zurich Research Center

***Abstract*—Agentic Large Language Models (LLMs) are rapidly becoming an important tool for architecture research, raising a concrete question for the computer architecture community: can an architect simply ask an agent to design a novel mechanism with a better score (*e.g.*, IPC) and expect a meaningful answer? In this work, we study two modes of agentic use on the same microarchitectural domain with the same agentic model: *unguided*, in which the agent receives a problem statement, a curated seed library, a scoring function, and constraints, and *guided*, in which a human domain expert additionally reviews iterations and redirects the search when it stalls or drifts.**

Our evaluation shows that the unguided agent produces a design that performs close to, but below, the state-of-the-art, while the guided agentic design outperforms it. From this contrast, we distill a practical recipe for using today's agents in microarchitecture research: they are most effective not as autonomous architects but as high-throughput collaborators whose output improves substantially under a human architect's steering.

I. Introduction

COMPUTER architecture research is an iterative search process. Architects propose schemes, implement and evaluate them in simulators, analyze their behavior, and refine them repeatedly until they mature. Modern architectural design spaces are enormous, making this process slow and labor-intensive, and often leaving substantial performance untapped.

Recent advances in agentic LLMs have made it tempting to ask whether this bottleneck can be alleviated. Agents can read and modify code, execute experiments, analyze outputs, and iteratively refine candidate solutions. Prior work on LLM-driven algorithm discovery [1] and autonomous systems research [2] has shown that agents can generate competitive solutions. These results naturally raise the question of whether the same paradigm can transfer to computer architecture research.

A natural question follows: *can an agent autonomously discover a better microarchitectural mechanism?* Recent work suggests it can, under the right conditions. Sankaralingam [3] argues that the design space is too large for manual exploration. ArchAgent [4] autonomously discovers cache replacement policies while Agentic Architect [5] explores replacement, data prefetching, and branch prediction. Yet in each case, a human first fixes the search envelope (seed designs, scoring function, simulator interface, workload set) and the agent then searches autonomously within it. However, in prior art the human first sets the stage, then leaves it. In this work, we ask what we believe is the practically decisive question: holding the search envelope fixed, how much does it matter whether a human architect stays in the loop, i.e., reviewing and steering the iterations. To answer this question, we use a strong agent [6] and compare two modes: 1) an *unguided* setting in which the agent independently explores the design space and 2) a *guided* setting in which a human architect iteratively steers the search process using architectural insight and feedback from simulation results. We instantiate this comparison on instruction cache (L1i) prefetching, a mature domain with strong state-of-the-art, where further gains are hard-won.

Our case study points to an important lesson. Autonomous agents can reach or beat the state-of-the-art: [4] does so for cache replacement with no human in the loop; yet the reported gains are modest. In our case study, the unguided agent assembles a plausible design but fails to beat the state-of-the-art; with iterative expert guidance the same agent (same model, seed library, budget) outperforms both the unguided design and the state-of-the-art. Our claim is comparative: holding the search envelope fixed, adding an in-loop architect improves the agent's output. The lesson we draw is therefore methodological. Today's agents are most effective not as autonomous architects but as high-throughput collaborators, whose output improves substantially when a human architect structures, constrains, audits, and critically redirects the search between iterations. The primary contribution of this work is thus not a new L1i prefetcher, but this insight into how agentic LLMs should be used in computer architecture research. From our study, we derive three key findings:

- **Agents are amplifiers, not oracles.** The effectiveness of an agentic search process is fundamentally bounded by the quality of the human-defined search space, evaluation methodology, and iterative feedback.
- **Guided agents outperform unguided agents.** In the L1i prefetching context, the guided design outperforms the state-of-the-art, while the unguided design approaches but does not improve performance over the state-of-the-art.
- **The architect's role shifts from implementation to orchestration.** Agentic workflows reduce the manual effort of implementing and iterating on candidate schemes, but architects remain essential for framing the problem, selecting seeds, interpreting results, and catching flawed hardware assumptions, overfitting, or proxy-metric gains that lack real architectural benefit.

II. Modes of Agentic Use

We compare two modes of applying an agent to the same microarchitectural problem. Both share the same search envelope: model [6], prompt, seed library, scoring function, search budget, and workloads. They differ only in the human's role during the search. In the unguided mode, the agent searches autonomously: it proposes and implements candidate designs and the fixed harness scores them, with the best-scoring design taken as the result. In the guided mode, a human architect additionally reviews the search and redirects the next iteration.

Both modes apply the same one-time realism audit as a terminal accept/reject gate on the final design; because this audit is performed after the search completes, it can reject

- **Your goal:** design a novel mechanism in domain **D**.
- **Simulation infrastructure:** Use the **Y** simulation infrastructure for evaluation. Refer to `howto.txt` for instructions on how to use it. Use **Z** and **W** million instructions for warmup and simulation, respectively.
- **Prior art:** `/PriorArt` contains previously proposed mechanisms in domain **D** -- you can use them for inspiration but the final design should be novel. Each mechanism's directory contains a readme.txt and a paper.pdf that explain its design and operation.
- **Applications:** `/workloads/train` contains all the training workloads that you will use for this project.
- **Objectives/Tasks:**
  1. Improve over the state-of-the-art scheme in `/PriorArt` using **S** as the scoring function.
  2. Your storage budget must not exceed that of the state-of-the-art by more than **B** bytes.
  3. Your implementation must be realistic for hardware designs. Avoid algorithms that cannot be realized in hardware.
  4. Produce a complete C++ implementation of the new mechanism.

Fig. 1: Prompt used in both unguided and guided modes. For the study of Section III, D=L1i prefetching, Y=ChampSim [7], Z=20, W=20, S=IPC, and B=20480 (20KB).

a finalist but cannot change what was searched. The two modes therefore differ only in in-loop steering: in the unguided mode, intermediate selection is fully automated, whereas in the guided mode the human architect inspects the search and redirects it. In both modes, the agent may modify only the candidate's source file; the simulator, harness, scoring function, and workloads are read-only. The generated design has access to only runtime signals without any access to trace-level information. Section III-A describes the experimental setup, including the model, simulator, and applications.

The goal of this comparison is to quantify the gap between fully autonomous agentic designs and collaborative human–agent designs. The following sections describe the two modes.

A. Unguided Mode

The agent receives the prompt of Fig. 1 with a curated library of prior work in the target domain (papers, code), selected by the human architect as a reasonable starting point. Its task is to design a novel scheme that outperforms the state-of-the-art under a predefined scoring function. The agent is free to reuse, combine, or deviate from the provided material.

The agent iterates autonomously for a fixed time budget. After each iteration, it receives simulator feedback verbatim and independently decides how to revise its implementation. No human architect reviews intermediate outputs or intervenes during the process. This models a realistic out-of-the-box usage scenario in which a researcher applies an agent to an architectural problem without iterative expert steering.

The prompt of Fig. 1 imposes two constraints to ensure meaningful hardware-oriented designs. First, the additional storage budget must remain comparable to the state-of-the-art, since simply scaling storage is not itself a novel contribution. Second, the generated design must remain implementable within realistic hardware constraints rather than relying on impractical algorithms. We impose these constraints to test whether agents can produce meaningful hardware-oriented designs; quantifying agentic impact under unrealistic storage or complexity assumptions is out of scope for this work.

B. Guided Mode

In this mode, the agent receives exactly the same prompt, seed library, time budget, and constraints as in the unguided mode, presented in Section II-A. The only difference is that a human architect reviews the simulator output after iterations, diagnoses the causes of low-performance designs (*e.g.*, over-aggressive prefetching) and, based on the observed workload behavior, steers subsequent iterations.

CPU Core	4GHz, 352-entry ROB, 4-wide, hashed perceptron branch predictor
ITLB/DTLB/L2-TLB	64/128/2048-entry, 1/1/8-cycle, 8/8/16-entry MSHR
L1i/L1d	48KB, 5-cycle, 8/16-entry MSHR , EPI / BERTI
L2C	1.25MB, 10-cycle, 32-entry MSHR, SMS
LLC (per core)	2MB, 20-cycle, 64-entry MSHR
DRAM	8GB, 3200MT/s

TABLE I: Baseline configuration.

III. Case Study: Instruction Prefetching

To evaluate what in-loop human guidance adds over a fully autonomous agent, we study prefetching for instruction caches (L1i), a mature domain with strong prior art [8].

A. Experimental Setup

We use ChampSim [7], configured as shown in Table I. For workloads, we use emerging server workloads from Google [9], with 10 workloads used for training and a disjoint held-out set reserved for final evaluation. Note that training refers to the agent's search phase, in which candidates are scored on these 10 workloads; no model weights are updated and the LLM is fixed for all experiments of this work.

Agent and search budget. We use Claude [6]. Both unguided and guided modes receive the same prompt, iteration count, and time budget (48 hours). For each candidate, the evaluation harness automatically compiles the generated implementation, executes the training workloads, and returns simulator statistics and score to the agent.

Seed library. The agent receives the source code and papers of all IPC-1 L1i prefetchers [8], together with additional L1i prefetching work. The material serves only as a reference; the agent may reuse, modify, combine, or ignore prior designs. Under our infrastructure and baseline, EPI [10] is the state-of-the-art reference throughout this work and is included in the seed library for both modes (unguided, guided).

Scoring and constraints. The optimization target is geomean IPC across the training workloads (Fig. 1). Candidate designs are constrained to remain within 20KB of the state-of-the-art; designs exceeding the budget receive a failing score so the search proceeds to other candidates without human input. The budget is computed automatically by summing the bit cost of all prefetcher state, including in-cache metadata. The agent edits only the L1i prefetcher source file behind the simulator's prefetch hooks, which expose the instruction address stream; it cannot modify the simulator, harness, or workloads. Hardware-realism constraints (*e.g.*, no unbounded structures) are specified in the prompt and verified by a terminal human review of the final design, performed after the search has completed, as explained in Section II. In this study, the reported design passed this review: bounded structures within budget without unrealistic assumptions or simulator exploits. A finalist rejected at this review is replaced by the next-highest-scoring design from the completed search rather than by re-running it; this case did not arise.

B. The Unguided Design

The unguided agent converges on a hybrid L1i prefetcher, **MaGikaRP**, composed of a spatial Footprint Table (FT) and a temporal Distance-Indexed Table (DKT). FT is indexed by 4KB instruction regions and stores a 64-bit bitmap recording which of the region's cache lines were previously demanded;

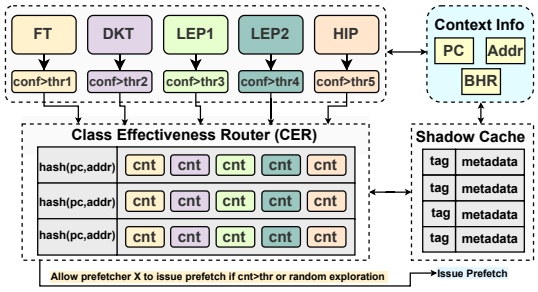

Fig. 2: Design of GyaRaDos.

the bitmap is replayed as prefetches on region revisit. DKT is indexed by line address and prefetches lines accessed K L1i references later, with $K \in \{2, 6, 12, 20\}$ implemented as parallel sub-tables. FT and DKT issue prefetches independently whenever their confidence exceeds a threshold. A small Shadow Cache (SC) mirrors the L1i tag array and records whether prefetched lines were demand-touched before eviction. The SC provides only a binary useful/useless feedback signal, a limitation later addressed by the guided design.

C. The Guided Design

The guided design, **GyaRaDos**, emerges through three human-guided interventions triggered by failures observed during the agent's iterations toward a new L1i prefetcher. Fig. 2 presents its design and operation.

1) Latency-entangled prediction and IP chaining. The human architect notes that EPI times prefetches by observed miss latency, which the unguided DKT design cannot capture. Therefore, the agent adds two Latency-Entangled Pair (LEP) tables linking the line in flight during a miss with the line that later fills. LEP1 is indexed by the line address, while LEP2 is indexed by a folded $(PC \oplus BHR)$ signature. The agent also adds HIP, a PC-indexed table storing up to four successor lines per indexing PC, each with its own confidence counter [11].

2) Online arbitration. With FT, DKT, LEP1, LEP2, and HIP issuing prefetches simultaneously, bandwidth pressure rises and IPC regresses on many workloads. The agent introduces the Class Effectiveness Router (CER), a per-context (PC, address hash) table of 4-bit effectiveness counters, one per prefetch engine. A prefetcher may issue requests only when its counter exceeds a threshold. A $1/16$ random-exploration probability allows any prefetcher to fire regardless of counter state, preventing permanent lockout in cold contexts.

3) Outcome-to-source feedback. CER converges slowly because usefulness outcomes cannot be attributed to the responsible prefetcher. The agent extends MaGikaRP's Shadow Cache (SC) by attaching to prefetched lines a 32-bit producer tag encoding its originating prefetcher state. Evictions update the corresponding CER entry and prefetcher confidence, converting the SC from a binary usefulness tracker into a per-prefetcher feedback scheme that closes CER's feedback loop.

All interventions stayed within the agent's reachable design space. The first supplied search direction: the architect pointed to a mechanism class (latency-based timing). The other two supplied failure attribution: the agent saw the IPC regressions but could not trace their structural causes, so each was a diagnosis from workload-level behavior (beyond what the agent could infer from aggregate scores) followed by a redirection.

D. Performance Evaluation

Fig. 3 compares MaGikaRP and GyaRaDos against EPI across both the training and held-out workloads. Results are normalized to a baseline with EPI and reported both as IPC speedup and the composite metric of Eq. 1.

IPC is our performance metric, but effective L1i prefetching also depends on coverage, cache pressure, and timeliness. We fold these metrics in a composite metric, used only as a post-hoc diagnostic rather than a search objective; the agent uses IPC as the optimization metric (Section III-A, Fig. 1).

$$\text{CompMetric} = R_{\text{IPC}}^2 \left(1 + Cov_{\text{L1i}}\right) R_{\text{MPKI}}^{1/2} R_{latency} \quad (1)$$

Here, $R_{\text{IPC}} = \text{IPC}_{pref}/\text{IPC}_{base}$, Cov_{L1i} is prefetch coverage, $R_{\text{MPKI}} = \text{LLC MPKI}_{base}/\text{LLC MPKI}_{pref}$, $R_{latency} = lat_{base}^{\text{L1i}}/lat_{pref}^{\text{L1i}}$. The IPC term is squared to prioritize performance. Ratio terms are set to 1 when the corresponding metric is not meaningful (*e.g.*, negligible LLC MPKI).

Without human intervention, the agent converges on a functional design, MaGikaRP, that performs slightly worse than the state-of-the-art. Specifically, MaGikaRP approaches EPI on many workloads, trailing by only 0.7% in geomean speedup across all workloads. The same trends hold on both training and held-out workloads. Although MaGikaRP scores higher on the composite metric of Eq. 1, this reflects coverage and timeliness rather than end-to-end performance, which is the optimized objective (Fig. 1). The composite metric is never used to rank designs. The unguided design ultimately plateaus because 1) DKT cannot align prefetch timing with variable fetch latency, leading to late prefetches on long-latency control-flow transitions and 2) FT and DKT issue prefetches independently, generating excessive prefetch traffic that increases LLC pressure and cache pollution. Although the agent observes the resulting IPC regressions, it fails to identify their structural causes without human guidance.

The guided design, GyaRaDos, overcomes these limitations through the three interventions described in Section III-C. Across all workloads (similar trends for training and held-out), GyaRaDos improves geomean IPC by 0.8% over EPI and by 1.5% over MaGikaRP. It also achieves a 30% improvement in the composite metric of Eq. 1 over EPI, as shown in Fig. 3, indicating that its performance gains are accompanied by higher cache efficiency and better bandwidth utilization. Notably, GyaRaDos outperforms or matches MaGikaRP on IPC (the optimized objective) across all workloads. The unguided design leads on a few workloads only under the composite metric, which neither mode optimizes.

The largest gains occur on workloads with deep front-end stalls and irregular control flow, such as `yankee`, where LEP1 and LEP2 achieve high miss coverage and good timeliness by exploiting long fetch-latency windows to learn source-effect timing. Conversely, high-MPKI workloads such as `sierra` inflate prefetch volume; here CER throttles low-accuracy engines and uses the most accurate ones, avoiding over-prefetching and LLC pollution seen in the unguided design.

The broader lesson is methodological. The same agent, given the same seed library, prompt, simulator, scoring function, and budget, produces qualitatively different outcomes

Fig. 3: Per-workload speedup (top) and composite metric (bottom) of the unguided and guided designs over the EPI baseline, for training and test workloads; boxplots (right) summarize each distribution. IPC speedup (top) is the optimized objective (Fig. 1); the composite metric (bottom) is a post-hoc diagnostic, not a ranking criterion.

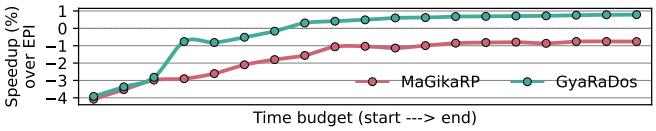

Fig. 4: Geomean speedup over EPI versus time budget.

depending on whether a human architect participates in the search loop. The unguided agent assembles competent mechanisms from prior ideas and approaches the state-of-the-art. The guided workflow enables the agent to cross that boundary by restructuring the search space and preventing convergence to a locally optimal solution. In this sense, *today's agentic LLMs behave less like autonomous researchers and more like high-bandwidth architectural collaborators*.

1) Search Progression: Fig. 4 shows how both designs improve over the search budget (same for both modes). The two curves coincide until the first human intervention; thereafter MaGikaRP plateaus without outperforming EPI, while GyaRaDos, redirected by the human architect (Section III-C) continues improving and finally outperforms EPI.

IV. A Practical Recipe

This section distills our case study into a practical recipe.

1) Treat agents as high-throughput collaborators, not autonomous architects. Agents rapidly explore the design space while architects provide direction and validation. Effectiveness depends less on autonomous discovery and more on how human expertise structures and steers the search process.

2) Choose the seed library carefully. Seeding the search with strong prior art worked well in our study. Prior art shows that weaker or more diverse seeds can encourage broader exploration [3], [12]; our strong-seed choice may have limited our gains, so we frame this as experience, not prescription.

3) Design the scoring function carefully. In our study, IPC alone sufficed for GyaRaDos to improve over the state-of-the-art. Other metrics may help in other domains, so the scoring function is a domain-dependent choice, not a fixed default.

4) Curate training workloads. Diversity matters more than volume: a small, diverse training set generalizes better than a large one dominated by similar workloads. Over-representing one workload class biases the signal and yields brittle designs.

5) Steer iteratively. Current agents explore design variations well but are weak at diagnosing root causes from aggregate metrics. The human architect's inspection of simulation outcomes and workload behavior is what closes this gap, enabling targeted corrections that provide better designs.

6) Audit implementations for realism. Improved scores do not guarantee valid designs. Candidates must be inspected for unrealistic assumptions. Constraints should be enforced explicitly and violations should lead to rejection.

V. Conclusions

In our study, the gap between the unguided and the guided agentic designs stems from two limitations of autonomous search: 1) *search direction*: while the agent can implement arbitrary schemes, identifying promising regions of the design space remains difficult without human guidance and 2) *failure attribution*: agents optimize against aggregate metrics, whereas architects can diagnose the root causes of regressions. We provide early evidence isolating the value of keeping a human architect in the loop: with the search envelope held fixed, the architect's role shifts from low-level implementation to structuring and steering the search: defining the problem, curating seed designs and workloads, constructing evaluation metrics, guiding iterative exploration, and auditing implementations.

Our study targets a single domain, only one instance of a broader class of microarchitectural problems to which agentic workflows apply. Whether weak search direction and failure attribution are fundamental to frontier agents or artifacts of today's models remains open; for now, these two axes are where a human architect adds the most value, and where autonomous agents should improve. The key question is therefore not whether agents will become fully autonomous architecture researchers, but how human architectural expertise can most effectively guide and refine agentic design exploration.

References

[1] Novikov *et al.*, "AlphaEvolve: A coding agent for scientific and algorithmic discovery," *arXiv:2506.13131*.

[2] A. Cheng *et al.*, "Barbarians at the gate: How AI is upending systems research," *arXiv:2510.06189*.

[3] Sankaralingam, "Computer architecture's AlphaZero moment: Automated discovery in an encircled world," *arXiv:2604.03312*.

[4] Gupta *et al.*, "ArchAgent: Agentic AI-driven computer architecture discovery."

[5] Blasberg *et al.*, "Agentic architect: An agentic AI framework for architecture design exploration and optimization," *arXiv:2604.25083*.

[6] Anthropic, "Claude," https://www.anthropic.com, 2025.

[7] Gober *et al.*, "The championship simulator: Architectural simulation for education and competition," *arXiv:2210.14324*.

[8] "1st Instruction Prefetching Championship," https://research.ece.ncsu.edu/ipc/.

[9] "4th Data Prefetching Championship," https://sites.google.com/view/dpc4-2026/.

[10] Ros *et al.*, "A cost-effective entangling prefetcher for instructions," in *ISCA'21*.

[11] Vavouliotis *et al.*, "Morrigan: A Composite Instruction TLB Prefetcher," in *MICRO'21*.

[12] A. Sharma, "OpenEvolve: an open-source evolutionary coding agent," https://github.com/algorithmicsuperintelligence/openevolve, 2025.