Agentic Architect: An Agentic AI Framework for Architecture Design Exploration and Optimization

Alexander Blasberg
Carnegie Mellon University
Pittsburgh, PA 15213
ablasber@andrew.cmu.edu

Vasilis Kypriotis
Carnegie Mellon University
Pittsburgh, PA 15213
vkypriot@andrew.cmu.edu

Dimitrios Skarlatos
Carnegie Mellon University
Pittsburgh, PA 15213
dskarlat@cs.cmu.edu

Abstract

Designing high-performance microarchitectural policies has historically required years of expert effort over vast combinatorial design spaces. Recent advances in LLM-driven code evolution suggest a new approach: pair a language model with cycle-accurate simulation and let evolutionary search produce candidate designs. We present Agentic Architect, an end-to-end framework that does this for three core CPU mechanisms: cache replacement, data prefetching, and branch prediction. Evolved policies match or exceed published state of the art policies in all three domains: $1.062\times$ geomean IPC over LRU (above Mockingjay's $1.056\times$), $1.76\times$ over no prefetching (vs. $1.59\times$ for VA/AMPM Lite and $1.55\times$ for SMS), and $1.100\times$ over Bimodal (above Hashed Perceptron at $1.085\times$). The human-defined search envelope (seed, scoring function, training traces, and prompt) drives the search far more than the choice of LLM or evolutionary backend, and evolved policies follow a consistent learned ensemble pattern: preserve the seed's core, add orthogonal predictors, and coordinate them at runtime.

1 Introduction

With the slowdown of Moore's Law, microarchitecture design has become a key bottleneck in processor and accelerator development. The traditional workflow asks an architect to design a policy, implement it, simulate it, inspect results, and iterate. This loop scales poorly to modern designs, whose interacting signals and control logic produce an exploration space too large to cover by hand.

Recent advances in agentic AI [Novikov et al., 2025, Romera-Paredes et al., 2024, Sharma, 2025, Cemri et al., 2026] pair LLMs with evolutionary search and automated evaluation, demonstrating AI-driven discovery across science and engineering. Computer architecture is a particularly suitable target: policies are naturally expressed as modular code, the design space is large but well-structured, and the field has mature evaluation infrastructure built around cycle-accurate simulators and standard benchmark suites [Bucek et al., 2018]. The central question we ask is whether LLMs can augment microarchitectural design-space exploration to produce designs competitive with the state of the art, and what role the human architect should play.

We introduce Agentic Architect, an agentic framework that places an LLM inside an evolutionary loop with a cycle-accurate simulator as the evaluator. The architect specifies the domain interface, seed policy, scoring function, hardware constraints, and training/evaluation trace split; the agent mutates candidate implementations, compiles and simulates them, and selects higher-scoring designs.

We evaluate Agentic Architect on three microarchitectural mechanisms spanning different optimization regimes: cache replacement, data prefetching, and branch prediction. In all three, evolved policies match or exceed strong baselines (Section 3). Our analysis shows that the human-defined search envelope (seed, scoring function, training traces, and prompt) drives results far more than the choice of LLM or evolutionary backend, and that evolved policies follow a consistent *learned ensemble*

AI Agents for Discovery in the Wild (AID-Wild), Workshop at ACM CAIS 2026.

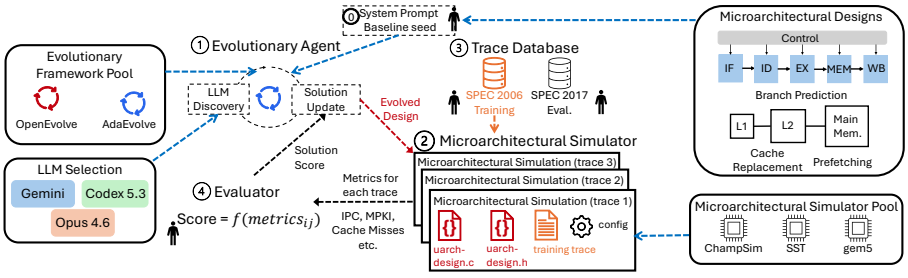

Figure 1: Overview of the Agentic Architect framework and the evolution loop. Human icons denote inputs provided by the architect: system prompt, seed policy, scoring function, and trace database with its training/evaluation split.

pattern: preserve the seed's core mechanism, add orthogonal predictive features, and coordinate them at runtime. To our knowledge, Agentic Architect is the first end-to-end framework for LLM-driven microarchitectural design optimization. The framework will be open sourced upon publication.

2 Agentic Architect

Agentic Architect (Figure 1) turns LLM-driven code evolution into a microarchitectural design loop. At each iteration, the evolutionary agent selects a parent design and prompts the LLM to generate a mutated candidate. The candidate is compiled and linked against a cycle-accurate simulator, run on the training traces, and scored through the evaluator's fitness function. Compilation failures and per-candidate simulation timeouts are caught early, assigned a sentinel score, and surfaced to the LLM in subsequent prompts to reduce repeated failures.

The framework is modular across evolutionary backends (we use OpenEvolve [Sharma, 2025] and AdaEvolve [Cemri et al., 2026]), LLMs (Claude Opus 4.6 [Anthropic, 2025] by default), and simulators that expose a stable policy interface (we use ChampSim [Gober et al., 2022]). Each domain is defined by a small fixed set of hooks (e.g., `find_victim` for replacement, `prefetcher_cache_operate` for prefetching, `predict_branch` for branch prediction), inside which the LLM writes arbitrary logic but cannot modify the interface itself.

Agentic Architect is best understood as a co-design framework: the architect defines the search envelope through four choices: the **seed policy** (the algorithmic foundation evolution refines), the **scoring function** (what the search rewards), the **training traces** (the workload distribution evolution sees), and the **prompt** (the diversity and reliability of generated candidates). As shown in Section 3, these choices dominate the choice of LLM or evolutionary backend.

Use cases and setup. We instantiate Agentic Architect on three domains. Cache replacement is seeded with Mockingjay [Shah et al., 2022] and compared to LRU, SHiP++ [Wu et al., 2011], Hawkeye [Jain and Lin, 2016], and Glider [Shi et al., 2019]. Prefetching is seeded with VA/AMPM Lite [Ishii et al., 2009] and compared to SMS [Somogyi et al., 2006], IPCP [Pakalapati and Panda, 2020], Pythia [Bera et al., 2021], and Berti [Navarro-Torres et al., 2022]. Branch prediction is seeded with the Hashed Perceptron [Jiménez, 2014] and compared to Bimodal. Cache replacement and prefetching use a fitness function of $\text{IPC} \times 10^4 - \text{LLC\_misses}/10^3$, and branch prediction uses $\text{IPC} \times 10^4 - 200 \cdot \text{MPKI}$. All experiments use ChampSim with a 4 GHz, 4-wide core (32 KB L1 / 256 KB L2 / 2 MB 16-way LLC, DDR5-3200) for memory experiments, and a 6-wide core with a 352-entry ROB and 20-cycle misprediction penalty for branch experiments. We evaluate on 11 SPEC CPU 2006/2017 traces with 25 M warmup followed by 50 M simulation instructions, and each evolution runs for 100 iterations.

3 Evaluation

Figure 2 summarizes results across the three domains; for each, we start from the SOTA seed and report the best result across OpenEvolve and AdaEvolve.

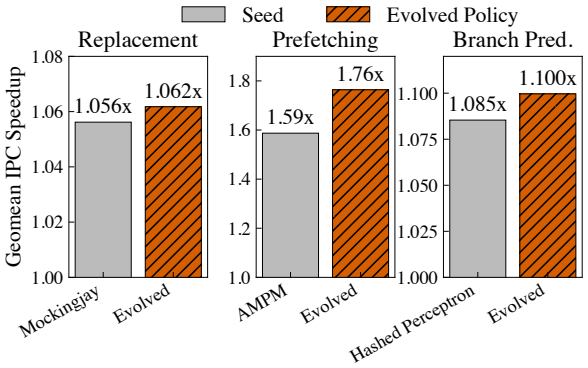

Figure 2: Geomean IPC speedup across the three domains vs state-of-the-art.

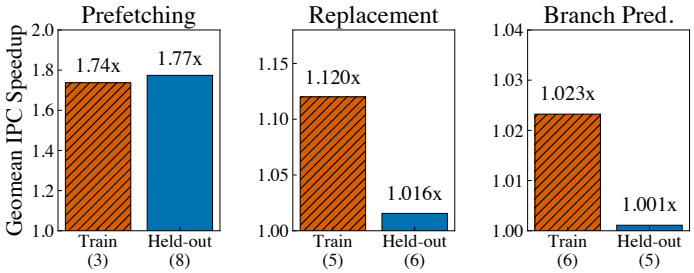

Figure 3: Training vs. held-out trace geomean IPC speedup.

Cache replacement. The best evolved policy achieves $1.062\times$ geomean IPC speedup over LRU, 0.6% above Mockingjay ($1.056\times$). Gains concentrate on memory-intensive traces ($1.49\times$ on `xalancbmk`, $1.12\times$ on `mcf`), with no regression on compute-bound traces. The evolved design preserves Mockingjay's reuse-distance predictor and adds orthogonal predictors and runtime arbitration.

Prefetching. The evolved prefetcher achieves $1.76\times$ over no prefetching, 17% above its VA/AMPM Lite seed ($1.59\times$) and 21% above SMS ($1.55\times$). Starting from VA/AMPM Lite's single per-page access map, evolution produced a six-engine architecture coordinated by an epoch-based self-tuner that re-evaluates each engine's accuracy every 256 accesses and disables underperformers.

Branch prediction. This is our most constrained domain, as Hashed Perceptron already exceeds 97% accuracy. The best evolved predictor achieves $1.100\times$ over Bimodal, 1.5% above the Hashed Perceptron seed ($1.085\times$), driven by `gobmk` (IPC $1.031 \rightarrow 1.182$, MPKI $8.53 \rightarrow 5.20$). The evolved design rediscovers and combines TAGE-SC-L [Seznec, 2011] and multiperspective-perceptron [Jiménez, 2016] components.

3.1 Generalization and Trace Selection

A central question is whether evolved policies generalize beyond their training set. Figure 3 compares each best policy on its training subset against held-out traces. **Prefetching shows remarkable generalization**, it was trained on three traces (`gcc`, `mcf`, `lbm`) spanning stride, pointer-chasing, and streaming patterns, and its held-out geomean ($1.77\times$) *exceeds* its training geomean ($1.74\times$). The runtime-adaptive six-engine architecture composes naturally with held-out workloads, which combine the same fundamental access patterns.

Replacement and branch prediction concentrate gains on training traces ($1.13\times$ vs. $1.004\times$ for replacement; $1.023\times$ vs. $1.001\times$ for branch), driven by single-trace specialization (`mcf`/`xalancbmk`; `gobmk`). We see that trace selection is the major factor governing generalization. For cache replacement, swapping `gcc` for `xalancbmk` in the training set *hurts* the 11-trace benchmark performance by 4%: `xalancbmk` has a 54% IPC spread across replacement policies, and its outsized fitness

contribution causes evolution to overspecialize. We see then that diversity of access patterns therefore matters more than coverage of workloads.

The seed and prompt show the same pattern. A SHiP-seeded run reaches only $1.050\times$ over LRU compared to our $1.062\times$ from Mockingjay, tracking the seed gap, meaning evolution amplifies a strong seed but cannot overcome a weak one. Minimal prompts also beat comprehensive prompts in both replacement ($1.053\times$ vs. $1.049\times$) and branch prediction ($1.100\times$ vs. $1.096\times$). Named techniques anchor the LLM to known architectures and often induce expensive logic that times out.

4 Discussion & Future Work

Across all three domains, evolved policies exhibit a consistent *learned ensemble* pattern. They preserve the seed mechanism, Mockingjay's reuse-distance predictor, VA/AMPM Lite's per-page access map, or the perceptron-style weighted predictor, while adding orthogonal signals, arbitration, and runtime adaptation based on accuracy, coverage, or set-level behavior. Individual components often resemble known techniques; novelty lies in their combinations and coordination rules. This recurrence suggests that the LLM has internalized useful structural priors from architecture literature. Evolution does not invent new primitives but reveals *where* those priors pay off under workloads. This reframes the architect's role: instead of implementing candidates, the architect defines the search envelope (seed, objective, traces, and prompt) while the LLM scales exploration within it.

Agentic Architect opens several directions for future architecture research. First, it can extend beyond the three policies studied here to any mechanism expressible as code and evaluable through simulation, including memory scheduling, coherence, NoC routing, page placement, and accelerator mapping. Extending the framework to RTL or silicon-aware backends would further allow implementation constraints to enter the optimization loop directly.

Second, future versions can make evolution more rigorous by incorporating storage, area, power, timing, and verification costs into the fitness function. Robustness can also become an explicit objective, rewarding designs that generalize across held-out traces. Because many architectural effects arise from component interactions, future systems should co-evolve mechanisms, such as prefetching and cache replacement, rather than optimizing each policy in isolation.

A longer-term direction is to optimize the search process itself. Beyond evolving microarchitectural policies, the framework could propose scoring functions, prompt strategies, and trace-selection policies, with architects validating the resulting search structure. This points toward agentic co-design, where human judgment moves higher in the design stack while LLM agents scale exploration.

5 Related Work

LLM-driven algorithm discovery [Romera-Paredes et al., 2024, Novikov et al., 2025] establishes LLMs as effective code-mutation operators, and several open-source frameworks [Sharma, 2025, Cemri et al., 2026, ao et al., 2025, Lange et al., 2025, Liu et al., 2025, Wang et al., 2025] extend this paradigm; we use two of them; OpenEvolve and AlphaEvolve. Machine learning is increasingly applied across systems research, including learned indexes [Kraska et al., 2018], RL-based query optimization [Marcus et al., 2019], and AI-driven discovery for systems algorithms [Cheng et al., 2025]. Concurrent work [Gupta et al., 2026] explores cache replacement via AlphaEvolve. To our knowledge, Agentic Architect is the first end-to-end framework spanning seed design, prompt strategy, scoring-function design, LLM selection, evolutionary search, simulator integration, trace selection, and generalization analysis across cache replacement, branch prediction, and prefetching.

6 Conclusion

Agentic Architect matches or exceeds state-of-the-art microarchitectural designs across cache replacement, prefetching, and branch prediction by combining LLM-driven evolution with cycle-accurate simulation. The architect's role shifts from manual policy design to envelope definition: a co-design framework where the human shapes the search and the LLM scales exploration. The framework will also be released open source upon publication to support that broader exploration.

References

Anthropic. System Card: Claude Opus 4 & Claude Sonnet 4. Technical report, 2025. URL https://www.anthropic.com/transparency.

Henrique Assumpç ao, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: an Open Source Evolutionary Coding Agent for Algorithmic Discovery and Optimization. arXiv preprint arXiv:2510.14150, 2025. URL https://arxiv.org/abs/2510.14150.

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1121–1137, 2021. doi: 10.1145/3466752.3480114.

James Bucek, Klaus-Dieter Lange, and Jóakim von Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, 2018. doi: 10.1145/3185768.3185771.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. Adaevolve: Adaptive llm driven zeroth-order optimization, 2026. URL https://arxiv.org/abs/2602.20133.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, Jeff Chen, Lakshya Agrawal, Aditya Desai, Jiarong Xing, Koushik Sen, Matei Zaharia, and Ion Stoica. Barbarians at the Gate: How AI is Upending Systems Research. arXiv preprint arXiv:2510.06189, 2025. URL https://arxiv.org/abs/2510.06189.

Nathan Gober, Gino Chacon, Lei Wang, Paul V. Gratz, Daniel A. Jiménez, Elvira Teran, Seth Pugsley, and Jinchun Kim. The Championship Simulator: Architectural Simulation for Education and Competition. arXiv preprint arXiv:2210.14324, 2022. URL https://arxiv.org/abs/2210.14324.

Raghav Gupta, Akanksha Jain, Abraham Gonzalez, Alexander Novikov, Po-Sen Huang, Matej Balog, Marvin Eisenberger, Sergey Shirobokov, Ngân Vũ, Martin Dixon, Borivoje Nikolić, Parthasarathy Ranganathan, and Sagar Karandikar. Archagent: Agentic ai-driven computer architecture discovery, 2026. URL https://arxiv.org/abs/2602.22425.

Yasuo Ishii, Mary Inaba, and Kei Hiraki. Access Map Pattern Matching for Data Cache Prefetch. In *Proceedings of the 23rd International Conference on Supercomputing (ICS)*, pages 499–500, 2009. doi: 10.1145/1542275.1542349.

Akanksha Jain and Calvin Lin. Back to the future: Leveraging Belady's algorithm for improved cache replacement. In *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 78–89. IEEE, 2016. doi: 10.1109/ISCA.2016.17.

Daniel A. Jiménez. Strided Sampling Hashed Perceptron Predictor. JILP Special Issue on the 4th Championship Branch Prediction Competition (CBP-4), 2014. URL https://jilp.org/cbp2014/paper/DanielJimenez.pdf.

Daniel A. Jiménez. Multiperspective perceptron predictor. Proceedings of the 5th Championship Branch Prediction Competition (CBP-5), 2016. URL https://jilp.org/cbp2016/paper/DanielJimenez1.pdf.

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 489–504, 2018. doi: 10.1145/3183713.3196909.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards Open-Ended And Sample-Efficient Program Evolution. arXiv preprint arXiv:2509.19349, 2025. URL https://arxiv.org/abs/2509.19349.

Gang Liu, Yihan Zhu, Jie Chen, and Meng Jiang. Scientific Algorithm Discovery by Augmenting AlphaEvolve with Deep Research. arXiv preprint arXiv:2510.06056, 2025. URL https://arxiv.org/abs/2510.06056.

Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A Learned Query Optimizer. In *Proceedings of the VLDB Endowment*, volume 12, pages 1705–1718, 2019. doi: 10.14778/3342263.3342644.

Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibá nez, Víctor Vi nals Yúfera, and Alberto Ros. Berti: an Accurate Local-Delta Data Prefetcher. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022. doi: 10.1109/MICRO56248.2022.00072.

Alexander Novikov, Ngân V u, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery. arXiv preprint arXiv:2506.13131, 2025. URL https://arxiv.org/abs/2506.13131.

Samuel Pakalapati and Biswabandan Panda. Bouquet of Instruction Pointers: Instruction Pointer Classifier-based Spatial Hardware Prefetching. In *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020. doi: 10.1109/ISCA45697.2020.00021.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical Discoveries from Program Search with Large Language Models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6.

André Seznec. A new case for the TAGE branch predictor. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 117–127. ACM, 2011. doi: 10.1145/2155620.2155635.

Ishan Shah, Akanksha Jain, and Calvin Lin. Effective Mimicry of Belady's MIN Policy. In *Proceedings of the 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022. doi: 10.1109/HPCA53966.2022.00048.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. Applying Deep Learning to the Cache Replacement Problem. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 413–425, 2019. doi: 10.1145/3352460.3358319.

Stephen Somogyi, Thomas F. Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. Spatial Memory Streaming. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, pages 252–263, 2006. doi: 10.1109/ISCA.2006.38.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. ThetaEvolve: Test-time Learning on Open Problems. arXiv preprint arXiv:2511.23473, 2025. URL https://arxiv.org/abs/2511.23473.

Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely Jr., and Joel Emer. SHiP: Signature-based Hit Predictor for High Performance Caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 430–441, 2011. doi: 10.1145/2155620.2155671.