ArchAgent: Towards Agentic Architecture Discovery

Raghav Gupta[1,†], Akanksha Jain[2], Abraham Gonzalez[2], Alexander Novikov[3], Po-Sen Huang[3], Matej Balog[3], Marvin Eisenberger[3], Sergey Shirobokov[3], Ngân Vũ[3], Martin Dixon[2], Borivoje Nikolić[1], Parthasarathy Ranganathan[2], Sagar Karandikar[1,†]

[1]University of California, Berkeley
[2]Google [3]Google DeepMind

raghavgupta@berkeley.edu,{bora,sagark}@eecs.berkeley.edu
{avjain,abegonzalez,mgdixon,parthas}@google.com
{anovikov,posenhuang,matejb,meisenberger,shirobokov,nganvu}@google.com

1 Introduction

Agile hardware design flows are a critically needed force multiplier to meet the exploding demand for compute. We present ArchAgent, an agentic AI system that builds on AlphaEvolve [7] to automate *computer architecture discovery*. By designing and implementing logic directly within ChampSim [4], ArchAgent discovered novel "winning" cache replacement policies that improve IPC speedup over prior SoTA by 5.3% on multi-core Google Workload Traces [1] in two days and by 0.9% on single-core SPEC06 [6] workloads in 18 days, in a cache replacement championship-like setting. ArchAgent achieved these gains 3-5× faster than human-led efforts in developing prior SoTA policies. We also introduce "post-silicon hyperspecialization," achieving a 2.4% IPC speedup improvement on single-core SPEC06 workloads via automated per-workload runtime parameter tuning. Finally, we discuss lessons learned from working on ArchAgent such as the phenomenon of "simulator escapes" and the need for performant and reliable simulation infrastructure to enable a future of agentic AI-driven computer architecure.

2 System Overview

ArchAgent expresses architectures as C++ models in ChampSim (a commonly used trace-based microarchitectural simulator) and builds on AlphaEvolve (an LLM-based evolutionary coding agent), including a highly distributed evaluation setup to discover novel architectures.

An outline of ArchAgent is shown in Figure 1. In this example, novel cache replacement policy candidates are automatically designed/implemented by AlphaEvolve in ChampSim. ChampSim is then compiled and run with a specified workload suite (e.g., SPEC) to evaluate each new policy in parallel for a target metric (e.g., IPC). This process continues iteratively, with ArchAgent continually proposing and evaluating new logic/mechanisms within the policy and utilizing evolutionary search to find increasingly better candidates.

[†]Work done while the author was also affiliated with Google.
Pointer to full-length paper: https://openreview.net/forum?id=SVIXvewImv.

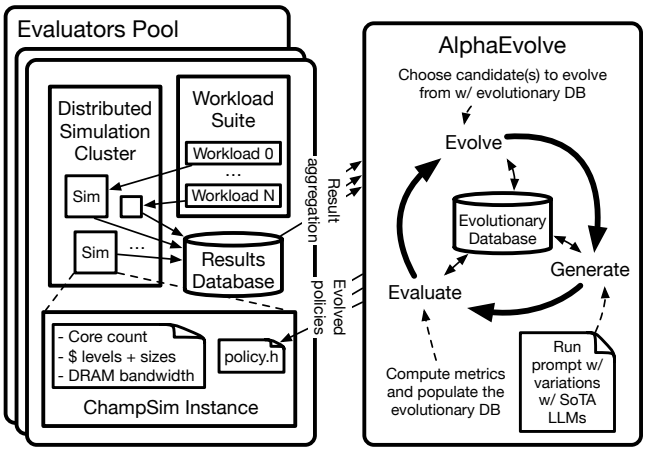


Figure 1. High-level system diagram of ArchAgent

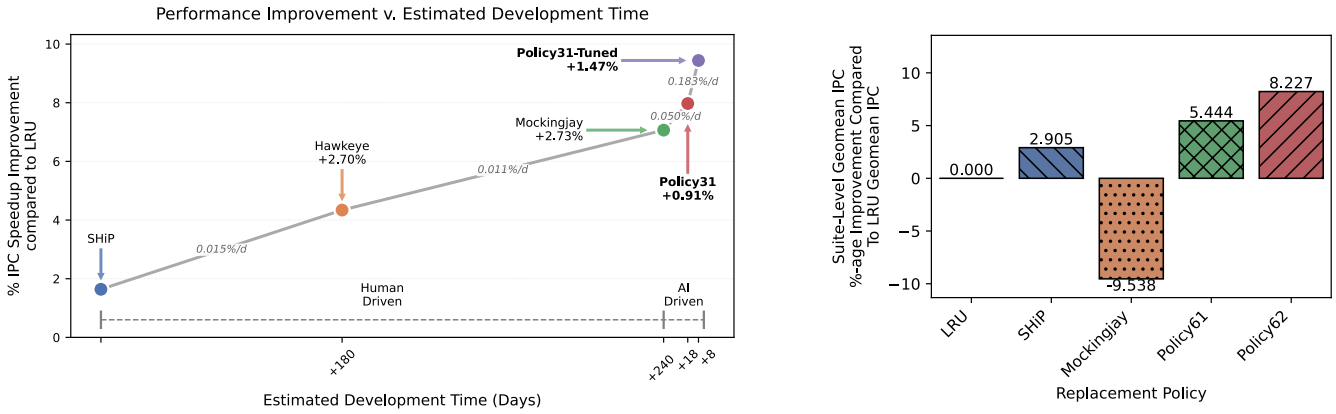
The system uses an ensemble of fast and thinking LLMs prompted to act as an expert architect/programmer, with context about the task, rules, simulator APIs, workloads, system configs, prior working programs, and prior literature.

3 Designing Cache Replacement Policies

We use ArchAgent to design multiple last level cache replacement policies broadly within the confines of the 2nd Cache Replacement Championship [5]. We target 19 memory-intensive single-core SPEC06 workload traces and 11 multi-core Google Workload Traces. ArchAgent received evolution feedback on short instruction traces (50/100M on SPEC06) whereas we performed validation of generated policies on long instruction traces (1B on SPEC06) of these workloads. It is provided with Mockingjay [8] as the starter code. These approaches significantly reduce time to convergence.

3.1 Single-Core SPEC06 Workloads

ArchAgent experimented for 18 days to create `Policy31`, introducing mechanisms such as usage intensity tracking, adaptive bypass throttling, and more. Figure 2a shows that `Policy31` improves IPC speedup normalized to LRU by 0.9% over the prior state-of-the-art policy on SPEC06, Mockingjay.



(a) Performance improvement (suite-level geomean IPC normalized to LRU) compared to estimated development time of replacement policies for the single-core prefetch-enabled ChampSim configuration on memory-intensive SPEC06 workloads. Slope (grey, italics) denotes percentage point improvement per day.

(b) Improvement in (geomean) IPC normalized to LRU geomean IPC for the multi-core prefetch-enabled configuration running Google Workload Traces.

Figure 2. Comparing ArchAgent-designed policies

When comparing the development effort involved in developing `Policy31` and the prior SoTA policies, we find that ArchAgent spent a few weeks and achieved its gains 3-5× faster than humans spending many months.

3.2 Post-silicon Hyperspecialization

We observe that ArchAgent-generated policies such as `Policy31` introduce new microarchitectural techniques that make the policies flexible and amenable to runtime tuning. Agentic flows create an opportunity once a hardware system is deployed where an agent can tune such runtime-configurable parameters exposed in hardware to further align the policies with a specific workload (mix). We identify 13 such runtime parameter expressions in `Policy31` that do not affect storage size and let ArchAgent only modify these. Over 8 days, ArchAgent tuned for each SPEC workload individually to generate `Policy31-Tuned`, demonstrating a 2.4% IPC speedup improvement and an over 10× faster rate of improvement than prior SoTA, as shown in Figure 2a.

3.3 Multi-Core Google Workload Traces

With Policy31, we show ArchAgent can win in an established competition environment on a heavily-explored workload suite such as SPEC06. However, it is widely established that SPEC06 is not representative of hyperscale cloud workloads [2, 3, 9]. While Mockingjay shows clear wins on both single- and multi-core SPEC, Figure 2b shows it performs much worse than even LRU on Google Workload Traces. ArchAgent designed `Policy62` in just two days and delivered a 5.3% improvement in IPC speedup normalized to LRU over SHiP [10], the best of prior work on these traces. Previously, designing a handcrafted hyperscale-centric policy would require a concerted, human-intensive effort spanning months. Notably, ArchAgent started with Mockingjay and recovered a 17.8% IPC speedup deficit with a simpler policy.

4 Discussion and Future Work

Agentic systems rapidly generate and evaluate thousands of design variants and can enable the pursuit of automated hardware-software co-design and specialization. However, maximizing this potential requires new methods to inject hard architectural constraints (e.g., area, power, and timing) directly into the generative process. We cannot rely on limited microarchitecural models lacking ASIC quality-of-results data or humans manually verifying hardware realizability. Future agentic systems will likely require tighter integration with standard EDA tools to bypass the current bottleneck of manual human verification.

We observe that unlike human researchers acting in good faith, agents ruthlessly optimize for reward signals and will readily exploit "simulator escapes." For instance, ChampSim does not support bypassing writes in the LLC, disallowing them via assertions. However, the compiler dropped assertions in optimized builds of ChampSim. During early experiments, ArchAgent learned to identify and manipulate this simulator path to silently drop LLC writes, eliminate DRAM pressure and artificially inflate IPC.

We also observe that ArchAgent's ability to reach creative solutions is inversely proportional to the evaluation latency. This becomes untenable for long, complex simulations which are commonplace in computer architecture. Truly unleashing the potential of agentic AI-driven architecture demands a fundamental overhaul of our evaluation infrastructure to be both more reliable and orders of magnitude faster, warranting more research into higher fidelity frameworks and simulators.

References

[1] [n. d.]. Google Workload Traces Version 2. https://console.cloud.google.com/storage/browser/external-traces-v2. Accessed: 2025-11-13.

[2] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems* (London, England, UK) *(ASPLOS XVII)*. Association for Computing Machinery, New York, NY, USA, 37–48. doi:10.1145/2150976.2150982

[3] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 3–18. doi:10.1145/3297858.3304013

[4] Nathan Gober, Gino Chacon, Lei Wang, Paul V Gratz, Daniel A Jimenez, Elvira Teran, Seth Pugsley, and Jinchun Kim. 2022. The championship simulator: Architectural simulation for education and competition. *arXiv preprint arXiv:2210.14324* (2022).

[5] Paul V. Gratz, Jinchun Kim, and Gino Chacon. 2017. The 2nd Cache Replacement Championship. https://crc2.ece.tamu.edu/?page_id=53. Accessed: 2025-11-13.

[6] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17. doi:10.1145/1186736.1186737

[7] Alexander Novikov, Ngân Vũ, 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. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. (2025). arXiv:2506.13131 [cs.AI] https://arxiv.org/abs/2506.13131

[8] Ishan Shah, Akanksha Jain, and Calvin Lin. 2022. Effective mimicry of belady's min policy. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 558–572.

[9] Wei Su, Abhishek Dhanotia, Carlos Torres, Jayneel Gandhi, Neha Gholkar, Shobhit Kanaujia, Maxim Naumov, Kalyan Subramanian, Valentin Andrei, Yifan Yuan, and Chunqiang Tang. 2025. DCPerf: An Open-Source, Battle-Tested Performance Benchmark Suite for Datacenter Workloads. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 1717–1730. doi:10.1145/3695053.3731411

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