Hunting for Offload: Automated Discovery of Acceleratable Code in Datacenters

Joshua Kim
The University of Texas at Austin

Chaojie Zhang
Microsoft Azure Research

Íñigo Goiri
Microsoft Azure Research

Christopher J. Rossbach
The University of Texas at Austin and Microsoft

Jovan Stojkovic
The University of Texas at Austin

Abstract

Modern datacenter CPUs integrate an expanding set of on-chip accelerators for system functions (e.g., compression) and ML inference (e.g., matrix units). Despite their availability, these accelerators see limited adoption in production software, as effective use requires extensive profiling, invasive refactoring, specialized programming models, and hardware expertise—costs that are particularly prohibitive for large, mature codebases. This lack of adoption wastes silicon capability and increases cost and energy consumption.

We argue that *agentic AI* is a necessary step toward making on-chip accelerators broadly *used* in datacenter software. However, accelerator enablement fundamentally differs from conventional CPU-centric optimization: it requires reasoning across heterogeneous and evolving hardware interfaces, handling input-dependent performance regimes, and operating with limited public tooling and training data.

We propose a research agenda that combines agentic-based program understanding with an augmented knowledge base to (1) identify code regions matching accelerator semantics, (2) map them to accelerator-backed implementations, and (3) determine when offloading is beneficial using input-sensitive runtime policies while preserving correctness. By framing accelerator adoption as an iterative, feedback-driven discovery rather than a one-shot compiler transformation, we show how agentic AI can unlock existing hardware capabilities for datacenter software at scale.

1 Motivation

Modern datacenter processors are rapidly becoming heterogeneous systems that pair general-purpose cores with an expanding set of on-chip accelerators to improve performance and energy efficiency for cloud workloads. These include engines for ubiquitous datacenter "tax" operations [4, 14, 16]—such as (de)compression, (de)encryption, data movement, and load balancing—as well as architectural extensions that enable AI/ML inference directly on CPUs, including AVX and AMX [2]. For example, recent Intel server generations (e.g., Sapphire Rapids [1]) integrate multiple on-die accelerator blocks [16], and major cloud providers are increasingly designing custom processors that similarly embed accelerators for datacenter services [12]. When effectively utilized, these units can reduce tail latency, increase throughput, and lower energy per request by offloading frequent, computationally intensive kernels from the CPU cores.

Despite this promise, accelerator adoption in production software remains the exception rather than the norm. Unlocking these capabilities often requires intrusive code refactoring, specialized programming models and toolchains, and deep, platform-specific expertise— dependencies that do not scale well to large, long-lived codebases with strict correctness and reliability requirements.

Hardware vendors such as Intel attempt to address this barrier by providing libraries and frameworks that encapsulate accelerator functionality behind higher-level APIs. In practice, these paths are often brittle: APIs evolve or deprecate, deployments face dependency and versioning conflicts, and the libraries are typically *static*, unable to reason about *when* offloading is beneficial given input size, data layout, contention, or system load. As a result, developers must still manually identify candidate regions, maintain accelerator-specific integrations, and tune invocation thresholds while preserving correctness and avoiding regressions.

The outcome is that today's datacenter software defaults to CPU execution paths, leaving expensive on-chip accelerators underutilized and turning shipped silicon into stranded capability [6]. This underutilization translates into higher infrastructure cost and exacerbates carbon footprint by deploying area and power that delivers little real-world benefit.

At the same time, recent advances in AI-assisted software engineering—such as LLM-based refactoring, performance tuning, and compiler optimizations for CPU systems [8, 9, 11, 15] and LLM-driven architectural reasoning [13]—demonstrate that AI can reason over program structure, correctness constraints, and performance behavior across complex stacks These results motivate extending AI-driven approaches toward the harder problem of automatically discovering and enabling effective on-chip acceleration.

2 Agentic AI for Accelerator Enablement: Challenges and Requirements

We argue that enabling effective use of on-chip accelerators is a natural target for AI-based optimization. However, it introduces substantially greater challenges than CPU-centric

tuning. Unlike conventional optimizations, accelerator enablement requires reasoning across heterogeneous and less standardized interfaces—spanning language runtimes, vendor libraries, ISA extensions, kernel drivers, and deployment constraints—where the "best" implementation may vary across microarchitectures and platform generations.

Moreover, accelerator performance and efficiency are often highly input-dependent: offload overheads, batching effects, cache and NUMA behavior, and contention for shared accelerator resources mean that benefits emerge only under certain payload sizes, data layouts, concurrency levels, or request mixes [3, 5, 7, 10, 16, 17].

Compounding these challenges, there is significantly less public knowledge, tooling maturity, and training data available for accelerator programming than for CPUs, leaving current models with far less prior exposure. Together, these factors make accelerator optimization difficult to capture with static compiler passes, hand-tuned heuristics, or off-the-shelf models, motivating *specialized agentic AI systems* that can integrate sparse domain knowledge, incorporate empirical feedback, and iteratively refine optimization decisions.

A key observation is that accelerator adoption is not a single transformation but a *closed-loop engineering process*. In practice, engineers must (i) discover candidate code regions, (ii) establish semantic equivalence under subtle corner cases, (iii) choose among multiple acceleration mappings (e.g., library calls vs. intrinsics vs. batching vs. format conversion), and (iv) continuously re-tune thresholds as software versions, inputs, and hardware platforms evolve. A static integration that helps on one request distribution can become neutral or harmful under different payloads, after a compiler upgrade, or when contention changes. This suggests that the core research challenge is to build systems that can *continuously* connect program semantics to available hardware capabilities and observed performance behavior, and update offloading decisions as conditions change.

3 An Agentic Framework for Automated Accelerator Offload

To that end, we envision *AccelAgent* as a multi-agent workflow that could automatically unlock on-chip accelerator usage in datacenter applications. Rather than treating acceleration as a one-off porting task, AccelAgent would frame it as an end-to-end transformation pipeline spanning (1) discovery of acceleratable regions, (2) rewriting and integration, (3) correctness validation, and (4) deployment-time adaptation.

At the front of the pipeline, AccelAgent would combine static analysis, profiling, and semantic reasoning to identify code regions that are strong candidates for hardware acceleration. When such regions are identified, the system would propose and apply code transformations that invoke the appropriate on-chip accelerators, either by substituting software implementations with accelerator APIs or by introducing architecture-specific intrinsics and compiler directives. We expect many opportunities to require *interface-level* changes (e.g., batching, format changes, or alignment fixes). An agentic system is well positioned to suggest such changes, justify them against program intent, and iteratively refine them when empirical feedback indicates mismatches. The transformed code would then be compiled and validated within the workflow to ensure functional correctness and semantic equivalence with the original implementation.

Beyond correctness, AccelAgent would explicitly reason about *when* offloading is beneficial. It would synthesize conditional invocation logic that dynamically selects between CPU and accelerator execution based on input-dependent characteristics such as payload size, data layout, compressibility, and concurrency, ensuring that accelerators are used only when they deliver performance or energy advantages. This policy-driven approach is essential in production: it converts acceleration from an all-or-nothing refactor into a robust, performance-aware mechanism that can avoid regressions and remain effective under workload drift.

Finally, we envision each stage being supported by specialized AI agents augmented with retrieval and measurement. These agents would draw on structured knowledge sources (e.g., vendor manuals, white papers, ABI details, and known pitfalls) and incorporate empirical profiling data (e.g., break-even points across problem sizes) to ground decisions in real behavior rather than solely in static reasoning. By coordinating these agents under a unified, feedback-driven workflow, AccelAgent represents a path toward making accelerator usage transparent, adaptive, and maintainable in production datacenter software.

4 Conclusion and Future Vision

In summary, while modern processors already include powerful on-chip accelerators, their benefits remain largely unrealized due to poor programmability and high adoption barriers. By leveraging agentic AI and a cooperative multi-agent design, our approach makes accelerator usage transparent, adaptive, and performance-aware, enabling datacenter software to fully exploit existing hardware capabilities. This can reduce cost and carbon footprint while improving performance, without requiring developers to manually reason about ever-growing hardware complexity.

Looking forward, we envision a broader ecosystem in which many specialized AI agents continuously cooperate across the datacenter stack—from application code and runtimes to compilers, OS, and firmware—to autonomously discover bottlenecks, synthesize safe optimizations, and deploy them with correctness and performance guarantees. Such a self-improving software layer could enable *self-evolvable* datacenter servers that adapt to changing workloads and new hardware generations with minimal human intervention.

References

[1] Intel. 2022. Technical Overview Of The 4th Gen Intel® Xeon® Scalable processor family. https://www.intel.com/content/www/us/en/developer/articles/technical/fourth-generation-xeon-scalable-family-overview.html.

[2] Intel. 2026. What Is Intel® Advanced Matrix Extensions (Intel® AMX)? https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-amx.html.

[3] Rishabh Iyer, Jiacheng Ma, Katerina Argyraki, George Candea, and Sylvia Ratnasamy. 2023. The Case for Performance Interfaces for Hardware Accelerators. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HotOS'23)*.

[4] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a warehouse-scale computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA'15)*.

[5] Jaeyoung Kang, Qirong Xia, Ipoom Jeong, Yongjoo Park, and Nam Sung Kim. 2025. Intel ® In-Memory Analytics Accelerator: Performance Characterization and Guidelines. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'25)*.

[6] Patrick Kennedy. 2022. Hands-on Benchmarking with Intel Sapphire Rapids Xeon Accelerators. https://www.servethehome.com/hands-on-with-intel-sapphire-rapids-xeon-accelerators-qct/7/.

[7] Reese Kuper, Ipoom Jeong, Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Jiayu Hu, Sanjay Kumar, Philip Lantz, and Nam Sung Kim. 2024. A Quantitative Analysis and Guidelines of Data Streaming Accelerator in Modern Intel Xeon Scalable Processors. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24)*.

[8] Ziyang Li, Saikat Dutta, and Mayur Naik. 2025. IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities. arXiv:2405.17238 [cs.CR] https://arxiv.org/abs/2405.17238

[9] Hannah Lin, Martin Maas, Maximilian Roquemore, Arman Hasanzadeh, Fred Lewis, Yusuf Simonson, Tzu-Wei Yang, Amir Yazdanbakhsh, Deniz Altinbüken, Florin Papa, Maggie Nolan Edmonds, Aditya Patil, Don Schwarz, Satish Chandra, Chris Kennelly, Milad Hashemi, and Parthasarathy Ranganathan. 2025. ECO: An LLM-Driven Efficient Code Optimizer for Warehouse Scale Computers. arXiv:2503.15669 [cs.SE] https://arxiv.org/abs/2503.15669

[10] Jiaqi Lou, Srikar Vanavasam, Yifan Yuan, Ren Wang, and Nam Sung Kim. 2025. Dynamic Load Balancer in Intel Xeon Scalable Processor: Performance Analyses, Enhancements, and Guidelines. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA'25)*.

[11] Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. 2025. SWE-fficiency: Can Language Models Optimize Real-World Repositories on Real Workloads? arXiv:2511.06090 [cs.SE] https://arxiv.org/abs/2511.06090

[12] Microsoft Azure. 2025. Announcing Cobalt 200: Azure's next cloud-native CPU. https://techcommunity.microsoft.com/blog/azureinfrastructureblog/announcing-cobalt-200-azure's-next-cloud-native-cpu/4469807.

[13] Shvetank Prakash, Andrew Cheng, Arya Tschand, Mark Mazumder, Varun Gohil, Jeffrey Ma, Jason Yik, Zishen Wan, Jessica Quaye, Elisavet Lydia Alvanaki, Avinash Kumar, Chandrashis Mazumdar, Tuhin Khare, Alexander Ingare, Ikechukwu Uchendu, Radhika Ghosal, Abhishek Tyagi, Chenyu Wang, Andrea Mattia Garavagno, Sarah Gu, Alice Guo, Grace Hur, Luca Carloni, Tushar Krishna, Ankita Nayak, Amir Yazdanbakhsh, and Vijay Janapa Reddi. 2025. QuArch: A Benchmark for Evaluating LLM Reasoning in Computer Architecture. arXiv:2510.22087 [cs.AR] https://arxiv.org/abs/2510.22087

[14] Akshitha Sriraman and Abhishek Dhanotia. 2020. Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*.

[15] Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025. SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution. arXiv:2502.18449 [cs.SE] https://arxiv.org/abs/2502.18449

[16] Yifan Yuan, Jiayu Hu, Ren Wang, Narayan Ranganathan, Reese Kuper, Ipoom Jeong, and Nam Sung Kim. 2023. On-chip Accelerators in 4th Gen Intel Xeon Scalable Processors: Features, Performance, Use Cases, and Future! ISCA'23 Tutorial.

[17] Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Sanjay Kumar, Philip Lantz, Vivekananthan Sanjeepan, Jorge Cabrera, Atul Kwatra, Rajesh Sankaran, Ipoom Jeong, and Nam Sung Kim. 2024. Intel Accelerators Ecosystem: An SoC-Oriented Perspective : Industry Product. In *Proceedings of the ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA'24)*.