Automated Discovery and Offloading of Acceleratable Code in Datacenters

1 Problem Statement

Modern datacenter applications adopt a microservice architecture that decomposes monolithic logic into independently deployable services, improving modularity and manageability, and has been widely adopted across many major IT companies [2,5,6,20,25–27]. However, this design incurs significant overhead from auxiliary "glue" operations, known as *datacenter tax*, such as (de)serialization for standardized data exchange [4] and (de)encryption for security, which collectively account for a large fraction of CPU cycles [11, 22, 23]. To mitigate this cost, specialized hardware accelerators have been developed for these operations [1,3,12,13,17], and are increasingly integrated into modern systems. For example, recent Intel Sapphire Rapids processors incorporate multiple on-die accelerators [10, 29], with similar designs adopted by cloud providers [21], enabling reduced tail latency and improved energy efficiency when effectively utilized.

However, despite the growing presence of on-chip accelerators in modern datacenters, they remain significantly underutilized. Programming accelerators requires deep expertise, invasive code refactoring, and familiarity with specialized programming models. Even more challenging is achieving *efficient* use, where numerous runtime factors (*e.g.* data size, arithmetic intensity) determine whether offloading provides the expected performance benefit. Collectively, these factors introduce substantial complexity, making it difficult for large codebases to integrate accelerators while preserving correctness and meeting performance goals.

Hardware vendors mitigate this complexity by providing high-level interfaces that abstract accelerator-specific details via libraries or shim layers [7, 8]. However, these abstractions are brittle: independent evolution of applications and libraries leads to versioning conflicts, and their static nature prevents runtime decisions on when offloading is beneficial. Moreover, applications often use multiple libraries, and naively composing vendor-accelerated components can introduce contention and interference, adding cross-component resource management burdens for developers.

As a result, datacenter applications continue to execute on CPU paths, leaving on-chip accelerators largely idle [14]. This leads to higher infrastructure costs, lost potential performance and efficiency, and inflated carbon footprint.

2 Proposed Solution

With the recent advances in AI, particularly in agentic code generation, a natural question is whether AI systems can bridge the gap in on-chip accelerator adoption. While promising, on-chip accelerators introduce challenges that fundamentally differ from conventional code optimization tasks.

2.1 Challenges in AI-Driven Accelerator Enablement

First, there is *high diversity in the set of accelerators*, each with distinct programming models, kernel drivers, and deployment constraints. Unlike CPU optimizations that operate within a unified ISA and runtime abstraction, accelerator usage requires reasoning across user-space libraries, kernel sub-systems, and device-specific configurations (e.g., virtual- or physical- function deployment, IOMMU). This diversity complicates the design of a unified AI-driven solution.

Second, the benefit of offloading to an accelerator is *highly sensitive to runtime parameters* such as data size and layout, batching, and concurrency-level. Figure 1 shows the tradeoff of offloading a `memcpy` operation to a data streaming accelerator (DSA [16]). We see that DSA outperforms CPU only beyond certain data sizes (*e.g.* 2MB), due to fixed offload overheads that dominate benefits from the accelerator.

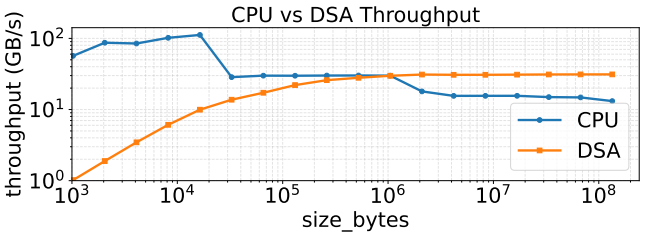

Figure 1: Throughput of `memcpy` on CPU vs DSA across input sizes.

Performance is not solely a function of input size. In many cases, achieving speedup requires restructuring software, for example, transitioning from synchronous to asynchronous execution models. To illustrate, we use `FFmpeg PNG` encoding as a running example. In this pipeline, each frame is passed to the encoder, which performs header construction and metadata formatting, followed by compression of the image data. Figure 2 shows performance when offloading compression to the QAT accelerator [9] under different software design models.

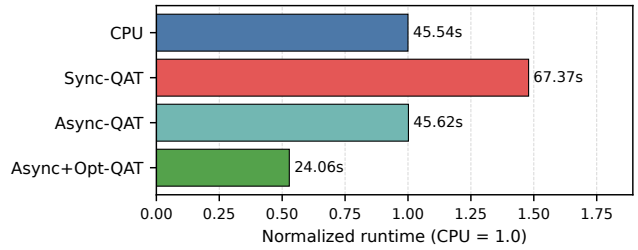

Figure 2: FFmpeg performance when offloading compression to QAT normalized to CPU

Following the original software structure yields a naïve synchronous per-frame offload that underperforms CPU execution due to blocking semantics and limited batching, resulting in a 1.48× slowdown. In contrast, asynchronous designs that expose concurrency improve throughput by 67.7%, reaching within 0.1% of CPU performance, and when coupled with further optimizations (batch sizing, polling intervals, stateful compression) achieve up to 1.89× speedup over CPU-only implementations. Overall, accelerator effectiveness depends on a multi-dimensional design space rather than simple heuristics.

Furthermore, datacenter applications are typically composed of *multiple interacting services, each with distinct performance objectives*. For example, throughput-oriented ser-

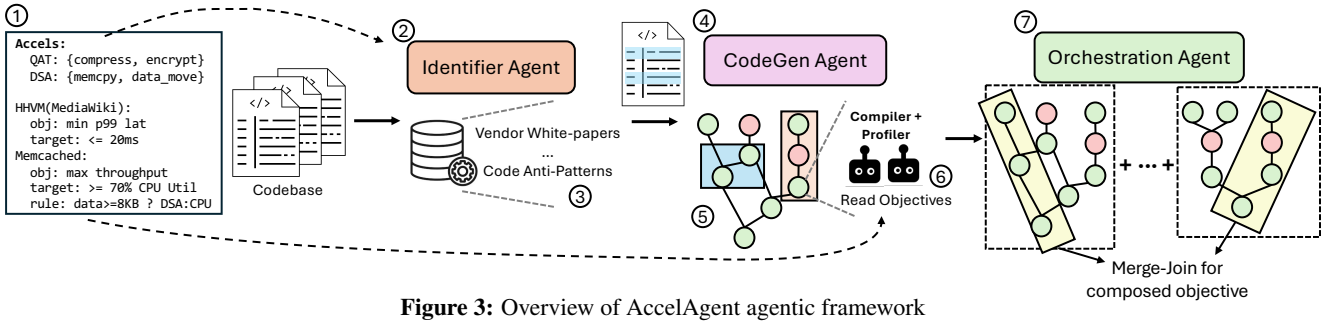

Figure 3: Overview of AccelAgent agentic framework

vices may benefit from large batching, while latency sensitive services may required fine-grained offloading. Therefore, optimizing accelerator usage within a single component is insufficient: independently optimized components may interfere when composed, leading to contention for shared accelerator resources, overall degrading application-wide performance.

Lastly, on-chip accelerator programming is limited by *lack of public knowledge, sparse documentation, and immature tooling ecosystems*. Unlike CPUs and GPUs, there is a gap in standardized training data that captures performance pitfalls or optimization strategies. This lack of structured knowledge presents a key challenge for AI-based approaches.

2.2 Overview of AccelAgent Framework

We propose *AccelAgent*, an agentic framework that automates the enabling of on-chip accelerators in datacenter applications. Figure 3 provides an overview of the system. AccelAgent decomposes the problem into four stages: (1) a developer facing specification interface, (2) identification of acceleratable code regions, (3) generation of candidate implementations, and (4) cross-service alignment and orchestration.

Developer Interface. AccelAgent exposes a DSL-like interface that allows developers to specify target accelerators and performance objectives (*e.g.* latency, throughput) for different services. As shown in (1), these specifications are provided alongside the application codebase. This abstraction enables developers to express intent without requiring detailed knowledge of accelerator programming models, while still constraining the search space to relevant optimization targets.

Identifier. The *Identifier Agent* ((2)) performs multi-pass analysis over the codebase to identify acceleratable regions for a target device. It detects code segments whose semantics match known accelerator capabilities (*e.g.* compression, encryption, data movement) and annotates them with mapping reasoning and implementation considerations. This process is augmented with external knowledge ((3)), including vendor documentation, usage examples, and pattern/anti-pattern snippets. Importantly, the agent distinguishes between *semantically* acceleratable regions and those that are practically beneficial to offload, filtering out cases where factors such as control flow divergence negate performance gains.

Code Generation. The *CodeGen Agent* ((4)) transforms annotated regions into accelerator-enabled implementations by exploring a structured design space via a *multi-objective candidate tree* ((5)). Each leaf in the tree represents a candidate optimized for a specific objective (*e.g.* throughput-oriented asynchronous execution). The agent may perform multiple passes to further optimize a candidate for the same objective, as shown in the orange highlight. Nodes can also be merged (blue highlight) to form parent candidates that combine multiple objectives. To ensure correctness and performance, each candidate is evaluated before further generation using compiler and profiler sub-agents ((6)). The whole process is guided by domain-specific resources, including programmer guides and sample accelerator code.

Orchestration. While the CodeGen Agent optimizes individual regions or services, the *Orchestration Agent* ((7)) coordinates accelerator usage across services. Given candidate trees for multiple services, the agent must resolve conflicting objectives. For example, a throughput-optimized batching strategy in one service may introduce latency violations for another. To address this, the agent defines a composed performance objective based on user-input. The agent then selects subtrees (yellow highlight) from each service that align with the global objective, even if they differ from locally optimal candidates. Then it performs a *merge-join* over candidate paths across services to identify candidate combinations that optimize overall application performance while reducing the search space.

Finally, the Orchestration Agent creates a cross-service execution plan, including work offloading and resource management policies to mitigate cross-component contention. This is validated via compiler and profiler sub-agents to ensure correctness and performance guarantees, even under composition.

3 Proposed Evaluation and Related Work

Evaluation. We will evaluate AccelAgent along two main axes. First, we will measure the accuracy of the Identifer agent using a curated set of benchmarks with both acceleratable regions and semantically-acceleratable but practically degrading regions. Second, we will evaluate end-to-end performance across applications from diverse domains, with a focus on datacenter applications (*e.g.* DCPerf [24]), capturing performance gains across increasing software complexity.

Related Work. Our work relates to prior efforts in automated CPU and GPU code optimization, including systems such as KernelEvolve [18] and ECO [19]. However, as previously described, on-chip accelerators introduce a new set of challenges, which our system specifically addresses. Additionally, there are hand-engineered accelerator integration works [15, 28]. AccelAgent does not attempt to find novel applications of the accelerators, but rather enable a large-scale adoption of on-chip accelerators in modern applications.

References

[1] Bulent Abali, Bart Blaner, John Reilly, Matthias Klein, Ashutosh Mishra, Craig B. Agricola, Bedri Sendir, Alper Buyuktosunoglu, Christian Jacobi, William J. Starke, Haren Myneni, and Charlie Wang. Data Compression Accelerator on IBM POWER9 and z15 Processors : Industrial Product. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, 2020.

[2] Amazon. What are Microservices? https://aws.amazon.com/microservices/, 2025.

[3] Junehyuk Boo, Yujin Chung, Eunjin Baek, Seongmin Na, Changsu Kim, and Jangwoo Kim. F4T: A Fast and Flexible FPGA-based Full-stack TCP Acceleration Framework. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23)*, 2023.

[4] Google. Protocol Buffers. https://protobuf.dev/, 2025.

[5] Google Cloud. What is Microservices Architecture? https://cloud.google.com/learn/what-is-microservices-architecture, 2025.

[6] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'23)*, 2023.

[7] Intel. Intel quickassist technology(qat) openssl engine (an openssl plug-in engine) which provides cryptographic acceleration for both hardware and optimized software using intel quickassist technology enabled intel platforms. https://github.com/intel/QAT_Engine, 2017.

[8] Intel. Qatzip: Compression library accelerated by intel® quickassist technology. https://github.com/intel/qatzip, 2017.

[9] Intel. Accelerate compression with intel® quickassist technology (intel® qat). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-qat.html, 2021.

[10] Intel. 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, 2022.

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

[12] Sagar Karandikar, Chris Leary, Chris Kennelly, Jerry Zhao, Dinesh Parimi, Borivoje Nikolic, Krste Asanovic, and Parthasarathy Ranganathan. A Hardware Accelerator for Protocol Buffers. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*, 2021.

[13] Sagar Karandikar, Aniruddha N. Udipi, Junsun Choi, Joonho Whangbo, Jerry Zhao, Svilen Kanev, Edwin Lim, Jyrki Alakuijala, Vrishab Madduri, Yakun Sophia Shao, Borivoje Nikolic, Krste Asanovic, and Parthasarathy Ranganathan. CDPU: Co-designing Compression and Decompression Processing Units for Hyperscale Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23)*, 2023.

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

[15] Hyungyo Kim, Qirong Xia, Jinghan Huang, Nachuan Wang, Younjoo Lee, Jung Ho Ahn, Wajdi K Feghali, Ren Wang, and Nam Sung Kim. Lilo: Harnessing the on-chip accelerators in intel cpus for compressed llm inference acceleration. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–17, 2026.

[16] Reese Kuper, Ipoom Jeong, Yifan Yuan, Ren Wang, Narayan Ranganathan, Nikhil Rao, Jiayu Hu, Sanjay Kumar, Philip Lantz, and Nam Sung Kim. 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)*, 2024.

[17] Nikita Lazarev, Shaojie Xiang, Neil Adit, Zhiru Zhang, and Christina Delimitrou. Dagger: efficient and fast rpcs in cloud microservices with near-memory reconfigurable nics. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[18] Gang Liao, Hongsen Qin, Ying Wang, Alicia Golden, Michael Kuchnik, Yavuz Yetim, Jia Jiunn Ang, Chunli Fu, Yihan He, Samuel Hsia, Zewei Jiang, Dianshi Li, Uladzimir Pashkevich, Varna Puvvada, Feng Shi, Matt Steiner, Ruichao Xiao, Nathan Yan, Xiayu Yu, Zhou Fang, Roman Levenstein, Kunming Ho, Haishan Zhu, Alec Hammond, Richard Li, Ajit Mathews, Kaustubh Gondkar, Abdul Zainul-Abedin, Ketan Singh, Hongtao Yu, Wenyuan Chi, Barney Huang, Sean Zhang, Noah Weller, Zach Marine, Wyatt Cook, Carole-Jean Wu, and Gaoxiang Liu. Kernelevolve: Scaling agentic kernel coding for heterogeneous ai accelerators at meta, 2026.

[19] 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. Eco: An llm-driven efficient code optimizer for warehouse scale computers, 2025.

[20] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'21)*, 2021.

[21] Microsoft Azure. 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, 2025.

[22] Akshitha Sriraman and Abhishek Dhanotia. 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)*, 2020.

[23] Jovan Stojkovic, Abraham Farrell, Christopher Hughes, Zhangxiaowen Gong, and Josep Torrellas. AccelFlow: Orchestrating an On-Package Ensemble of Fine-Grained Accelerators for Microservices. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'26)*, 2026.

[24] Wei Su, Abhishek Dhanotia, Carlos Torres, Jayneel Gandhi, Neha Gholkar, Shobhit Kanaujia, Maxim Naumov, Kalyan Subramanian, Valentin Andrei, Yifan Yuan, and Chunqiang Tang. 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)*, 2025.

[25] Think Software. Microservices Architecture of Twitter Service. https://thinksoftware.medium.com/design-twitter-microservices-architecture-of-twitter-service-996d 2021.

[26] Uber. Introducing Domain-Oriented Microservice Architecture. https://www.uber.com/blog/microservice-architecture/, 2020.

[27] Ketan Varshneya. Understanding design of microservices architecture at Netflix. https://www.techaheadcorp.com/blog/design-of-microservices-architecture-at-netflix/, 2021.

[28] Qirong Xia, Houxiang Ji, Yang Zhou, and Nam Sung Kim. Hardware-accelerated kernel-space memory compression using intel qat. *IEEE Computer Architecture Letters*, 24(1):57–60, 2025.

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