AKO: Agentic Kernel Optimization

Shuxiao Xie[1,2] **Shuyang Xie**[3] **Dezhi Ran**[1,4] **Wei Yang**[5] **Tao Xie**[1,2,4]
[1]Beijing Tongming Lake Information Technology Application Innovation Center (TLAIC), China
[2]Fudan University Institute of Systems for Advanced Computing, China
[3]Harbin Institute of Technology, China
[4]Key Lab of HCST (PKU), MOE; SCS, Peking University, Beijing, China
[5]University of Texas at Dallas, USA

Abstract

Production large language model (LLM) serving depends on hand-tuned graphics processing unit (GPU) kernels across a fast-growing matrix of operators and accelerators, and expert supply cannot cover it. Coding agents—a frontier model wrapped in a generic agent harness such as Claude Code (the agent loop, tool use, and skill mechanism it ships out of the box)—can now produce kernels that match or beat expert libraries. This moves the engineering question to the software layer around the agent: the *task-specific harness* that supplies the scoring, reference material, and orchestration a generic agent lacks for the task. We present AKO, such a harness around Claude Code, released as two open-source artifacts. AKO4ALL is a single drop-in skill that optimizes one kernel in place, with automatic multi-backend detection. AKO4X runs multi-round, per-operator campaigns in which fresh per-round sub agents share an accumulating archive of past kernels, coordinated manually or by a persistent master, with the benchmark reached through a swappable adapter (FlashInfer-Bench by default); it offers three modes—manual, automated, and an evidence-gated mode that accepts edits to its own skills only when backed by measured results. On NVIDIA B200 / Modal, across 10 FlashInfer-Bench operator targets AKO matches or beats the concurrent KDA submission on all five MLSys 2026 competition kernels (about $31\times$ over the FlashInfer expert on sparse attention, and above $1\times$ on the mixture-of-experts (MoE) and gated-delta-net (GDN) decode kernels, where KDA falls below the expert; the DeepSeek sparse-attention indexer has no runnable expert baseline, so the comparison there is head-to-head with KDA) and clears the expert baseline on four of five further targets, including an evidence-gated MLA paged-prefill campaign whose best kernel re-measures at $1.44\times$ the expert under the same CUDA 13.2 protocol.

`https://tongminglaic.github.io/AKO/`

1 Setting

Production LLM serving runs on a fast-moving operator-by-accelerator matrix; each cell needs a hand-tuned kernel (FlashInfer/cuDNN/FlashAttention-class expert work, expert-weeks each, redone every hardware generation). The current generation of coding-agent products—a frontier *model* wrapped in a *generic agent harness* (agent loop, context management, tool use, a skill mechanism), e.g. Claude Code—can drive long sequences of edits, tool calls, and measurements. Given such an agent, the model has enough capability to write competitive GPU kernels once the agent's environment is scoped for the task. This shifts the binding engineering work from the model to the *task-specific harness* layer around it: what tools the agent sees, what scoring is used, what is remembered across runs, and how a multi-round campaign is structured.

The layer split. We distinguish **(a)** the generic agent harness (Claude Code out of the box: agent loop, context, tool use, skill mechanism)—a fixed dependency we never modify—from **(b)** the task-specific harness: scoring, the baseline-freshness rule (re-measure the expert baseline on the current hardware), profiler integration, the SKILL catalog, task template, per-operator reference archive, master+sub orchestration, and the harness-edit ledger. AKO *is* layer (b); it does not train the model and does not modify the generic agent harness. This split is an interface boundary, not a claim that the layers are behaviorally independent—it lets us evaluate an upgraded model or generic harness with AKO held fixed, while when the task-specific harness should itself adapt to a stronger agent stays open. Its rationale is to put the kernel-specific feedback loop—scoring, profiling, reference material, memory, and orchestration—in layer (b), so an unmodified coding agent can apply its existing edit-and-tool-use capability to this vertical task.

2 System: AKO

AKO ships layer (b) as two artifacts. **AKO4ALL** packages it as a single Claude Code skill: drop a kernel into a directory and the skill detects the backend (CUDA / Triton / TileLang / CuTe DSL / HIP), bootstraps a flat KernelBench-style workspace, and loops profile $\rightarrow$ edit $\rightarrow$ bench $\rightarrow$ commit; no master/sub, no cross-session memory, no harness co-evolution—a true drop-in for one kernel (released 2026-03-24, MIT). **AKO4X** packages it as a template repository for multi-round campaigns. Because a single long-lived agent's context degrades across many attempts, each round is run by a fresh *sub* agent that inherits earlier rounds through a shared per-operator archive rather than a shared context window: a persistent *master* picks a parent from `reference/<family>/`, spawns a clean per-round environment (`spawn.py --operator`), and files the resulting variant back as the next round's candidate parent, while the sub draws capability on demand from nine SKILLs. All benchmark access goes through one adapter module, so swapping the scoring benchmark is a localized, fork-and-adapt edit—though its contract is derived from FlashInfer-Bench [FlashInfer Team, 2025] (the default), so ports adapt to AKO's FlashInfer-Bench data model and some coupling remains by design (one benchmark is active per checkout). Every harness edit in the evidence-gated mode (Mode 3, below) is logged to an append-only ledger.

Three modes (AKO4X). *(1) Manual sub-only*: the user runs a single sub through the inner loop and archives the result—the workflow behind most reported kernels. *(2) Automated master+sub* (default): the master drives N rounds, selecting parents and archiving variants, never editing the harness. *(3) Master+sub with opt-in harness co-evolution*: after each clean phase-1 optimization, the master resumes the same sub with a phase-2 retrospective; the sub proposes harness edits, and the master applies an evidence gate before accepting any into the shared harness. Comparability across rounds is held by a small frozen scope (task identity + scoring + baseline-freshness), so the harness that computes the reward cannot drift together with the reward.

The Mode-3 evidence gate. Because evaluation signals in kernel work are noisy and can be actively deceptive (hard-coded outputs, dropped precision, manipulated timing), an accepted edit needs more scrutiny than a raw speedup. The master applies a staged gate per proposal: a *scope* check (reject edits touching the frozen task identity, benchmark scoring/baseline, or the `master/` privilege boundary); an *evidence* check (reject unsupported or generic-knowledge proposals); SKILL-doc cleanup (strip session-specific notes before applying); folding related rules rather than blindly appending them; and *pre-archive integrity* checks that the motivating variant implements the operator itself (no vendor-kernel delegation) and matches no known cheat pattern (e.g. hard-coded outputs). Each decision is one append-only ledger line.

3 Experiments

We follow the official MLSys 2026 FlashInfer-Bench contest setup end-to-end: NVIDIA B200 (Blackwell, sm_100a) on Modal, with the contest-pinned stack (CUDA 13.2, "cu132"; PyTorch 2.12.0+cu132, Triton 3.6.0, FlashInfer 0.6.8.post1), evaluated under the official workload set and validator, using one off-the-shelf coding-agent product and model family (Claude Code, Claude Opus 4.6/4.7) with no task-specific training. Table 1 reports the five MLSys 2026 competition kernels head-to-head against the concurrent KDA submission [HAN Lab Kernel Mafia, 2026]; both use the same Claude Opus writer class (KDA adds an independent Codex verifier), so this is a

Table 1: Head-to-head on five MLSys 2026 FlashInfer-Bench kernels, both systems re-measured by us (B200 / cu132) vs. the same FlashInfer expert (mean over each kernel's official workload set, size n). †Indexer: the FlashInfer–DeepGEMM expert is unrunnable on cu132, so we report the latency head-to-head—AKO4X is $1.27\times$ faster than KDA (geomean per-workload ratio; both pass $128/128$).

Kernel	n	FlashInfer	KDA	AKO4X
DSA Sparse Attention	23	$1.00\times$	$4.6\times$	$\mathbf{31.1}\times$
DSA TopK Indexer†	128	—	—	$1.27\times$ vs KDA
MoE FP8 Block-Scale	19	$1.00\times$	$0.60\times$	$\mathbf{1.20}\times$
GDN Decode	54	$1.00\times$	$0.81\times$	$\mathbf{1.18}\times$
GDN Prefill	100	$1.00\times$	$1.72\times$	$\mathbf{2.37}\times$

systems-level comparison, not a model-training or component-ablation contrast. We additionally report five targets KDA does not cover, each reached with only `spawn.py --operator` and a brief archive seed: AKO4X clears the expert baseline on four (grouped-query attention (GQA) paged decode $1.46\times$, multi-head latent attention (MLA) paged decode $1.45\times$, MLA paged prefill $1.44\times$, RMSNorm $1.16\times$) and reports GEMM as an honest $1.00\times$ negative where no admissible DSL kernel beat cuBLAS.

Mode-3 case study. A 7-round `mla-paged-prefill` campaign (DeepSeek-V3 incremental prefill; B200, 2026-05-22) moved its best kernel from $1.40\times$ to $1.48\times$ the expert (consistent CUDA 13.2 re-measure $1.44\times$) and produced *five* accepted substantive harness edits to the triton / tilelang / cute-dsl SKILLs (e.g. a Triton warp-specialization rule, a TileLang numerical-hazard note, and a Blackwell tensor-memory (tmem) allocator section), each gated and logged. A second campaign on `rmsnorm-h128` accepted four further edits, exercising the same gate on a second operator (the evidence is two pilots, not a fully-scaled result); both ledgers ship in full.

Ranking note. AKO participated in the MLSys 2026 Full-Agent track but is absent from the official leaderboard for regional eligibility reasons; the credibility chain rests on the head-to-head table, breadth across 10 targets, and the public trajectory + reference archive + ledger.

4 Related work

We classify prior and concurrent work into four factual groups; head-to-head comparison lives above. **Pipeline-based** systems embed an LLM in a human-scripted procedure—role-fixed multi-agent (CudaForge, Astra, STARK) [Zhang et al., 2025b, Wei et al., 2025, Dong et al., 2025] or LLM-guided search with a learned heuristic inside a fixed procedure (AccelOpt [Zhang et al., 2025a], K-Search [Cao et al., 2026]). **Self-directed agentic** systems let the agent drive a static harness: AVO [Chen et al., 2025] (evolutionary variation operator; attention family on B200; agent not released), KDA [HAN Lab Kernel Mafia, 2026] (per-kernel phased prompt curriculum + Humanize loop with a Codex verifier; our head-to-head anchor), and AutoKernel [Jaber and Jaber, 2026] (fixed multi-stage playbook). **General-purpose harness-evolution** work—AHE [Lin et al., 2026a], Meta-Harness [Lee et al., 2026]—targets general coding benchmarks with an external verifier or held-out signal; AKO4X Mode 3 is the kernel-performance instance where the same harness that is edited also computes the reward. **Kernel-specialized model-training** (Kevin [Baronio et al., 2025], AutoTriton [Li et al., 2025], and related) is complementary: a stronger upstream model can be evaluated behind AKO with layer (b) held fixed, while whether the task-specific harness should then adapt remains future work.

Contributions. (1) AKO, a task-specific harness around Claude Code, released as two open-source artifacts (AKO4ALL, AKO4X). (2) An opt-in, evidence-gated harness-co-evolution mechanism (AKO4X Mode 3) with an append-only ledger. (3) Expert-competitive results on B200 / Modal across 10 FlashInfer-Bench operator targets, with a head-to-head against the concurrent KDA submission. (4) Two Mode-3 case studies with full public ledgers (`mla-paged-prefill`, `rmsnorm-h128`). Cross-benchmark generality (SOL-ExecBench [Lin et al., 2026b]) and a cross-GPU canonical sweep are future work.

References

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn rl for generating cuda kernels, 2025. URL https://arxiv.org/abs/2507.11948.

Shiyi Cao, Ziming Mao, Joseph E. Gonzalez, and Ion Stoica. K-search: Llm kernel generation via co-evolving intrinsic world model. http://arxiv.org/abs/2602.19128, 2026.

Terry Chen, Zhifan Ye, Bing Xu, Zihao Ye, Timmy Liu, Ali Hassani, Tianqi Chen, Andrew Kerr, Haicheng Wu, Yang Xu, Yu-Jung Chen, Hanfeng Chen, Aditya Kane, Ronny Krashinsky, Ming-Yu Liu, Vinod Grover, Luis Ceze, Roger Bringmann, John Tran, Wei Liu, Fung Xie, Michael Lightstone, and Humphrey Shi. AVO: Agentic variation operators for autonomous evolutionary search, 2025. URL https://arxiv.org/abs/2603.24517.

Juncheng Dong, Yang Yang, Tao Liu, Yang Wang, Feng Qi, Vahid Tarokh, Kaushik Rangadurai, and Shuang Yang. Stark: Strategic team of agents for refining kernels, 2025. URL https://arxiv.org/abs/2510.16996.

FlashInfer Team. flashinfer-bench: Benchmark infrastructure for flashinfer operators. https://github.com/flashinfer-ai/flashinfer-bench, 2025.

HAN Lab Kernel Mafia. Kernel design agents: HAN lab kernel mafia technical report, 2026. MLSys 2026 FlashInfer-Bench competition submission; manual archive, no arXiv ID; source https://github.com/mit-han-lab/mlsys2026-flashinfer-contest.

Jaber Jaber and Osama Jaber. AutoKernel: Autonomous GPU kernel optimization via iterative agent-driven search, 2026. URL https://arxiv.org/abs/2603.21331.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. http://arxiv.org/abs/2603.28052, 2026.

Shangzhan Li, Zefan Wang, Ye He, Yuxuan Li, Qi Shi, Jianling Li, Yonggang Hu, Wanxiang Che, Xu Han, Zhiyuan Liu, and Maosong Sun. Autotriton: Automatic triton programming with reinforcement learning in llms, 2025. URL https://arxiv.org/abs/2507.05687.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. http://arxiv.org/abs/2604.25850, 2026a.

Zihan Lin et al. Sol-execbench: Speed-of-light benchmarking for real-world gpu kernels against hardware limits, 2026b. URL https://arxiv.org/abs/2603.19173.

Anjiang Wei, Tianran Sun, Yogesh Seenichamy, Hang Song, Anne Ouyang, Azalia Mirhoseini, Ke Wang, and Alex Aiken. Astra: A multi-agent system for gpu kernel performance optimization, 2025. URL https://arxiv.org/abs/2509.07506.

Genghan Zhang, Shaowei Zhu, Anjiang Wei, Zhenyu Song, Allen Nie, Zhen Jia, Nandita Vijaykumar, Yida Wang, and Kunle Olukotun. Accelopt: A self-improving llm agentic system for ai accelerator kernel optimization, 2025a. URL https://arxiv.org/abs/2511.15915.

Zijian Zhang, Rong Wang, Shiyang Li, Yuebo Luo, Mingyi Hong, and Caiwen Ding. Cudaforge: An agent framework with hardware feedback for cuda kernel optimization, 2025b. URL https://arxiv.org/abs/2511.01884.