OPTIMASX: An Agentic End-to-End GPU Performance Optimization Workflow in the Code Editor

[1]Yugesh Bhattarai, [2]Jessica Rubin, [1]Prajan Manoj Kumar Rekha, [1]Mohammad Zaeed, [1]Tanzima Z. Islam
yugesh@txstate.edu, jessica.rubin@bison.howard.edu, prajan@txstate.edu, cup7@txstate.edu, tanzima@txstate.edu
[1]Texas State University, [2]Howard University

ABSTRACT

Optimizing a GPU kernel costs 30 manual steps before code changes: pick a profiler, reach a remote system, submit a job, read the report. Domain scientists hold the most GPU hours and the least profiling expertise, so no one tunes the code with the most to gain. OPTIMAS removes two steps by guiding a language model with runtime evidence, but assumes the user can produce it [1]. We transformed OPTIMAS into OPTIMASX, a multi-agentic system that produces the evidence and removes all 32 steps. From a VS Code extension, OPTIMASX profiles code, reads evidence, proposes changes, checks correctness, and measures runtime. The extension holds workflow state, so a queued job or backend restart cannot end the session. On an A100, OPTIMASX reached 4.80× on XSBench and 1.19× on Sobol with bit-identical output, and extends to Cholla and eight HeCBench apps. The speedups are OPTIMAS's, and the end-to-end automation is ours.

I. INTRODUCTION

A developer who wants a GPU speedup must obtain cluster access, configure the environment, write a batch script, pick a profiler, submit a job, and read the diagnostics.

Nsight Compute reports stall reasons and roofline position but stops before the code change. CUDA-L1 optimizes for measured speedup but never sees the diagnostics that explain the bottleneck [2].

OPTIMAS proved that diagnostics guide a model, not how a scientist obtains them, and reports that collecting them dominates end-to-end cost [1]. OPTIMAS runs with a fixed stage order. One process cannot outlive a queue, a fixed order cannot skip completed work, and a compiled pipeline cannot take a new profiler. We transformed OPTIMAS into a multi-agentic system, OPTIMASX: we gave order to a planner, each capability to a tool, and all state to the VS Code extension.

II. OUR APPROACH

The manual profiling barrier. Users start without diagnostics. We therefore made OPTIMASX check whether they exist. If not, the profiling agent prepares the files, adapts a submission template, submits the job, and watches it. If so, it reuses them and spends no second allocation.

The asynchronous execution barrier. A cluster job can wait hours. In that time the editor closes or the backend restarts. OPTIMAS runs its loop inside one live process, so if either happens, it loses the context it needs to collect the result. We therefore kept progress out of the process. The VS Code extension stores the objective, history, finished steps, and job identifiers, and sends them with each request. The backend rebuilds the workflow, takes the next step, returns the new state, and forgets, so one session spans the hours a cluster takes to answer.

TABLE I
MANUAL STEPS THE USER PERFORMS, BY STAGE. OPTIMAS REMOVES THE TWO OPTIMIZATION STEPS, AND OPTIMASX REMOVES ALL 32.

Stage	Fully manual	OPTIMAS	OPTIMASX
Profiling	15	15	0
Analysis	15	15	0
Optimization	2	0	0
Total	**32**	**30**	**0**

The fixed-pipeline barrier. OPTIMAS fixed its stage order, so each new capability required a developer to edit the source. We therefore replaced the pipeline with agents that call tools. Skill files tell each agent its tools, the preconditions that order them, and its limits. Each capability is a Model Context Protocol (MCP) tool, so a new profiling mode is just a script.

We gave the VS Code extension the conversation and the state, hosted the planner and its sub-agents on Google ADK, and their tools on an MCP server. A `gemini-3.1-flash-lite` planner delegates through `AgentTool` calls, so each agent returns to the planner rather than to another agent. The optimization agent calls `gemini-2.5-pro`, and the planner invokes it only after the user confirms.

Run-to-run variation. Because an LLM selects each tool, tool ordering is not guaranteed identical across repetitions of an identical prompt. A fixed seed, skill files that require profiling results before analysis, a per-turn cap on tool calls, and an output-match check limit this.

III. PRELIMINARY RESULTS

Table I gives the main result: OPTIMASX removes all 32 manual steps, leaving the user to state an objective and approve the edit. Without diagnostics, the agent submitted the job and returned its identifier. With uploaded files, it classified them

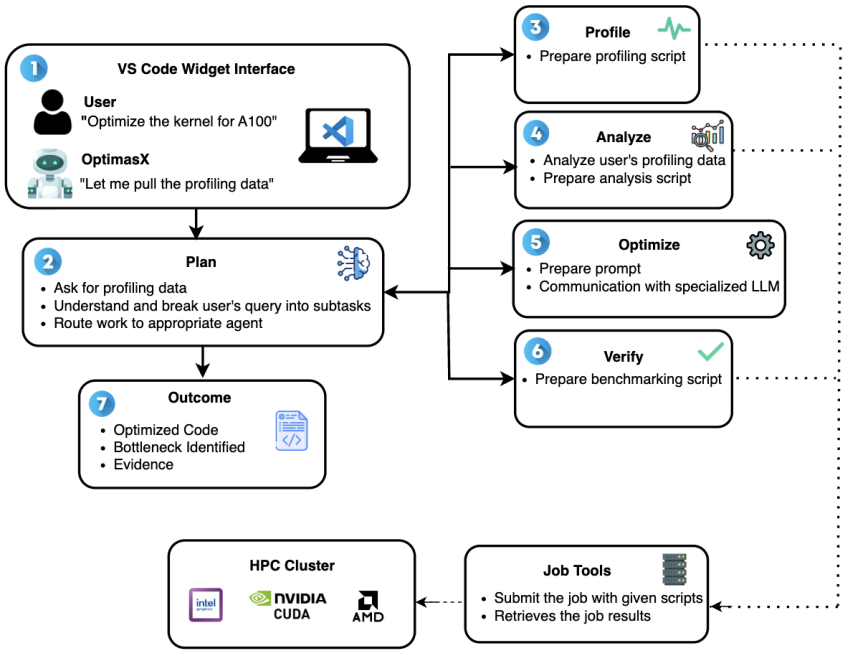


Fig. 1. OPTIMASX workflow. The user submits code and a target GPU. OPTIMASX plans, profiles, analyzes, optimizes, and verifies, then returns evidence-backed code. Existing profiling data is reused when available and collected on Lonestar6 when not.

TABLE II
SPEEDUP (×), MEAN ± STD. DEV. OVER FIVE RUNS ON AN A100. SPEEDUPS ARE INHERITED FROM OPTIMAS. HECBENCH: BOXFILTER, CHEMV, CONCAT, CONVOLUTIONSEPARABLE, ACCURACY, BINCOUNT, AMGMK, BLOCKACCESS.

Method	Sobol	XSBench	Cholla	HeCBench (8)
OPTIMAS	1.18 ± 0.11	4.83 ± 0.22	1.504 ± 0.001	2.282 ± 2.465
OPTIMASX	1.19 ± 0.11	4.80 ± 0.18	1.504 ± 0.000	3.17 ± 3.39

correctly. The session survived two injected backend restarts across five turns because the extension restored the state. We added three profiling modes, with no orchestration change.

We ran four benchmarks through the full chain from the editor, five runs each (Table II). OPTIMASX calls the same optimization core at the API default temperature rather than the 0.15 OPTIMAS selected, so the rows are not directly comparable. XSBench fell from 0.1391 s to 0.0290 s (4.80×) and Sobol from 3356.90 μs to 2809.49 μs (1.19×), both with bit-identical output. One profiling session cost $0.0019 in planner inference over two turns.

IV. IMPACT AND FUTURE WORK

OPTIMASX puts GPU optimization behind one request inside VS Code, where roughly 50 million developers already work [3]. A user reaches OPTIMAS, which improved runtime in 98.82% of 3,410 experiments [1]. New architectures and kernels can be added without modifying the system.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation under Grant No. 2443561 and Award No. CCF-2621730. The authors acknowledge TACC at The University of Texas at Austin. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Portions of the text were drafted and edited with assistance from Claude (Anthropic), and the authors reviewed all content.

REFERENCES

[1] M. Zaeed, T. Z. Islam, and V. Indic, "OPTIMAS: An intelligent analytics-informed generative AI framework for performance optimization," in *Proc. 32nd ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD '26)*, Jeju Island, Republic of Korea, Aug. 9–13, 2026, doi: 10.1145/3770855.3818188.

[2] X. Li, X. Sun, A. Wang, J. Li, and C. Shum, "CUDA-L1: Improving CUDA optimization via contrastive reinforcement learning," in *Proc. ICLR*, 2026. [Online]. Available: https://arxiv.org/abs/2507.14111

[3] A. Silver, "Celebrating 50 million developers: The journey of Visual Studio and Visual Studio Code," Microsoft for Developers Blog, May 2025. [Online]. Available: https://developer.microsoft.com/blog/celebrating-50-million-developers-the-journey-of-visual-studio-and-visual-studio-code/