PerfAdvisor

AI Agent for Diagnosing GPU Performance Profiles

Leon Hostetler

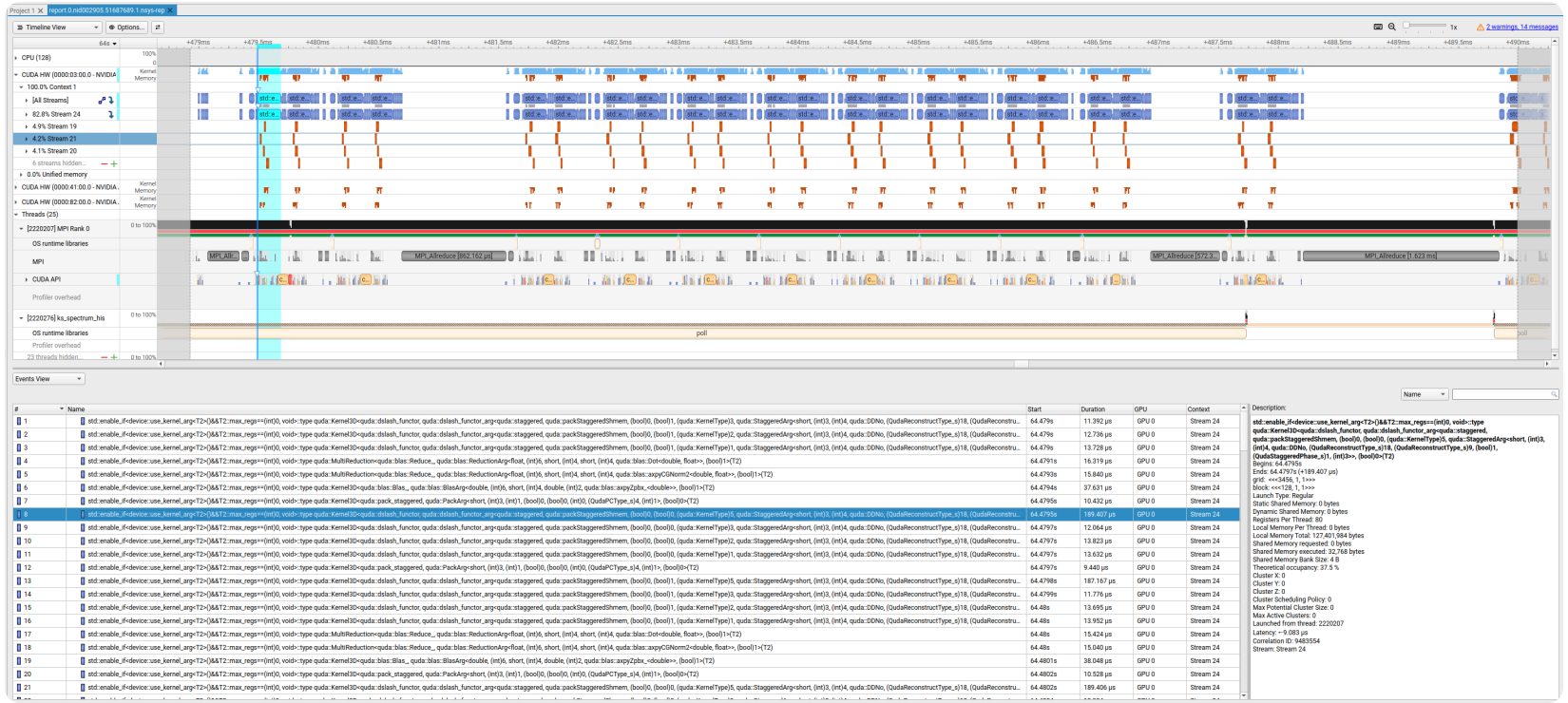
Department of Physics, Indiana University

July 27, 2026

Motivation

The problem

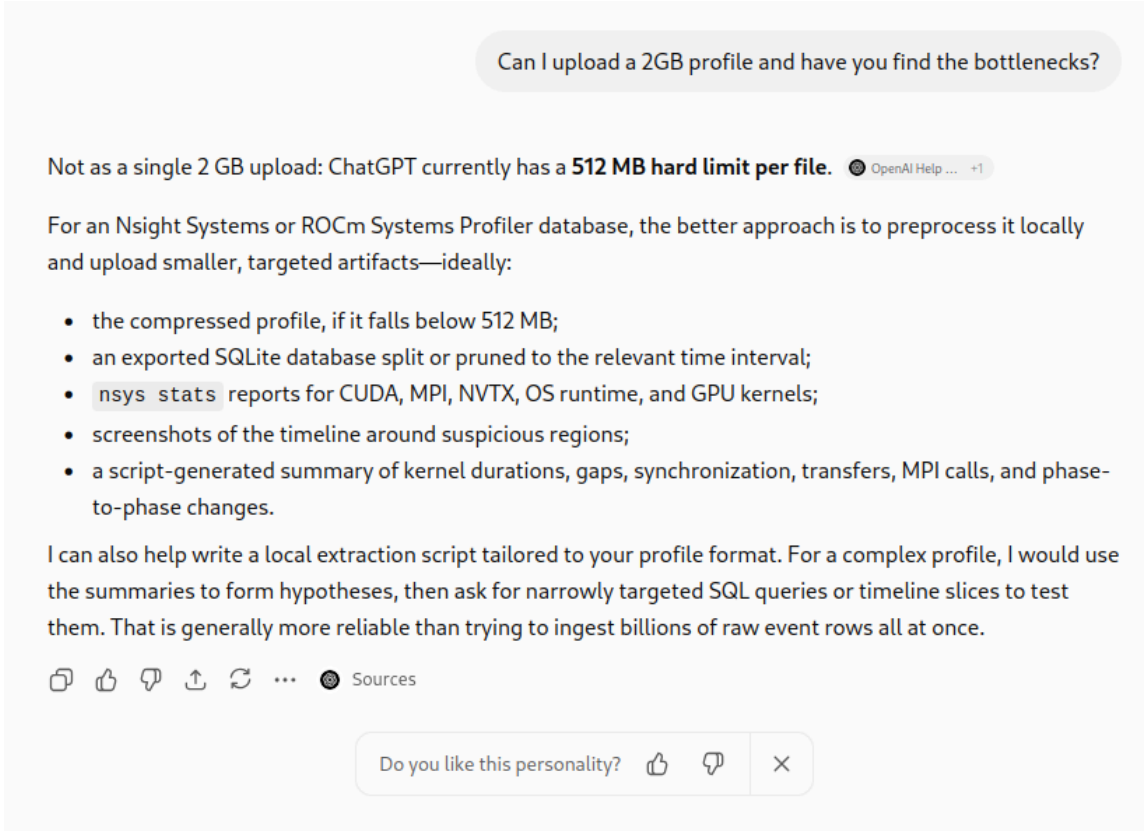
- GPU applications on supercomputers generate profiles with millions of events



- Finding a bottleneck means cross-correlating kernels, MPI events, transfers, and idle gaps, while having a good understanding of the hardware and network
- Lattice QCDers should not need to be GPU architects and HPC network engineers to get performant code
- Can we harness existing LLMs to help with this?

But how?

ChatGPT politely declines:



A familiar shift: from chatbot to agent

In two years, using an LLM for code went from a *chatbot you copy-paste with* to an *agent that does the work*.

Old: chatbot in a browser (you are the harness)

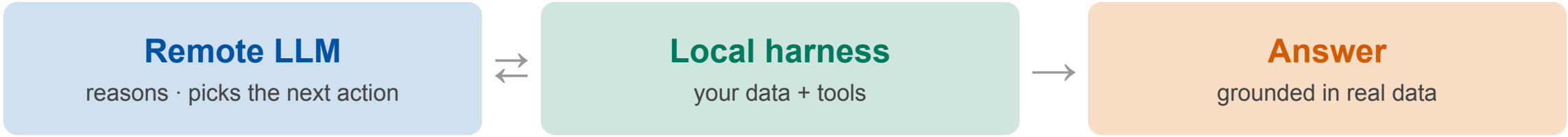
1. Paste code into ChatGPT, ask what's wrong
2. Paste the reply back into your editor
3. Recompile and test
4. Repeat

New: a coding agent (e.g. Claude Code)

State the goal; the agent runs the loop itself—reads the files, runs the compiler and tests, edits the code, reads the errors, and tries again until it works.

You set the goal; the agent drives the loop.

An agent = an LLM given tools and a loop: your data and tools stay *local*, the reasoning stays *remote*, and it iterates until the goal is met.



The same shift, for analyzing GPU profiles

A profile is millions of events in a SQLite database—analyzing it maps onto the same kind of loop.

Old: by hand and eye

- Open the profile in the vendor GUI, or query the SQLite tables yourself
- Cross-correlate kernels, MPI events, transfers, and idle gaps *by eye*
- Form a hypothesis, run another query, repeat
- Time: *Hours of work; GPU-architect expertise.*

New: an agentic harness

- Local: the profile database + analysis tools (kernel stats, gap histograms, MPI breakdown, raw SQL)
- Remote: the LLM reasons over the metrics, calls tools for the detail it needs, ranks hypotheses
- Time: *Minutes, with evidence-grounded bottlenecks and fixes.*

> ***PerfAdvisor is that harness for GPU profiles**—you bring the profile; the agent drives the investigation.*

What it does

PerfAdvisor is a lightweight Python package that harnesses an LLM to reason over profiles and quickly surface pathologies, identify bottlenecks, and suggest mitigations.

- Ingests *ROCm Systems Profiler* (AMD) and *Nsight Systems* (NVIDIA) profiles.
- Returns ranked, evidence-grounded hypotheses with concrete mitigations.
- Backends: Anthropic · OpenAI · Google Gemini.

Input: A profile database (or several, for multi-rank MPI runs)

Output: A ranked list of bottleneck hypotheses, each labeled with its execution phase, its supporting evidence, and a suggested fix.

Scope: PerfAdvisor is designed for *systems profiles* (i.e. big picture interplay between algorithm, hardware, and network) from *Nsight Systems* and *ROCm Systems Profiler*. It reads a *tracer's* view—API calls, kernel timing, transfers, MPI, idle gaps. Pathologies that can only be diagnosed with hardware counters (e.g. *Nsight Compute* / *rocprofv3*) are out of scope...not as a matter of principle but simply to keep the project manageable.

How it works

How to use PerfAdvisor

Step 1 — Profile your application

For `nsys` (e.g. Perlmutter):

```
srun -N 1 -n 8 nsys profile -t cuda,osrt,mpi --mpi-impl=mpich ./my_executable args
```

For `rocprof` (e.g. Frontier):

```
export ROCPROFSYS_USE_ROCPD=true
srun -N 1 -n 8 rocprof-sys-sample -- ./my_executable args
```

Step 2 — Run PerfAdvisor

```
export ANTHROPIC_API_KEY="sk-ant-apXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
perf-advisor analyze /profiles/rank_*.db --model=claude-opus-4-8
```

Pipeline overview

1. **Local Analysis** — profile database is read, a *phase-detection* algorithm segments the timeline into phases (init, main compute, teardown, …), then summary metrics (kernels, MPI, gaps,...) are computed per phase.
2. **Agent Loop** — initial prompt bundles the metrics, the tool catalog, and instructions, then the agent runs a multi-turn ReAct-style[1] loop until the LLM is satisfied or `--max-turns` is hit.
3. **Output** — ranked, evidence-grounded hypotheses with concrete mitigations.

[1] S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," *Proc. ICLR*, 2023.

Local Analysis: phase detection & per-phase metrics

1. PerfAdvisor segments the timeline into non-overlapping execution phases. Why is this necessary?
   i. A real-world profile has multiple distinct compute phases, e.g., load the lattice, gauge fix, CG solves, correlator tying, tear-down, etc. Behavior and bottlenecks are very different for different phases, and averaging across the whole thing would just muddle the picture
   ii. Now, behavior can be analyzed *per phase* and each hypothesis can be anchored to a particular phase

PerfAdvisor

Detected Execution Phases

Phase	Start (s)	End (s)	Duration (s)	GPU Util %	GPU Kernel (s)	GPU Idle (s)	Top Kernel	Top MPI Op
idle	0.0	90.725	90.725	0.0%	0.0	0.0	–	MPI_Barrier (34.439s)
Kernel2D	90.725	96.525	5.801	17.9%	1.036	4.766	Kernel2D (35.8%)	MPI_Wait (0.614s)
Kernel3D	96.525	141.119	44.594	72.0%	32.119	10.907	Kernel3D (31.0%)	MPI_Start (1.7s)

2. Summary metrics (GPU busy/idle, top kernels, dominant MPI ops, etc.) are computed *per execution phase*

The agent loop

The LLM doesn't see the raw profile—it pulls detail on demand by calling local tools: reason → call a tool → read the result → repeat, until it can commit to ranked hypotheses. Example of one turn:

PerfAdvisor

```
[local] Analyzing rank_683330_rocpd-0.db (provider=anthropic, model=claude-opus-4-6)

──────────────────────── Turn 1 / 20 ────────────────────────
[local] Context size ≈ 11,951 tokens
[← llm] Let me now gather more detailed data on memory copies, gaps, and kernel performance counters for the key
phases.
[← llm:tool] memcpy_summary({"start_ns": 524394844828476, "end_ns": 524439438422635})
[← llm:tool] gap_histogram({"start_ns": 524394844828476, "end_ns": 524439438422635})
[← llm:tool] mpi_summary({"start_ns": 524298319454022, "end_ns": 524389044243783})
[local] memcpy_summary → {"transfers": [{"kind": "Device-to-Device", "transfers": 155039, "total_bytes":
566787096576, "total_s": 27.2049, "effective_GBs": 20.83, "pct_of_peak_bandwidth": null}, {"kind": "Host-to-Device",
"tr...
[local] gap_histogram → {"total_idle_s": 10.907, "buckets": [{"label": "<10us", "count": 851, "total_s": 0.006},
{"label": "10-100us", "count": 40921, "total_s": 2.156}, {"label": "100us-1ms", "count": 36574, "total_s": 5.55...
[local] mpi_summary → {"mpi_present": true, "ops": [{"op": "MPI_Barrier", "calls": 221193, "total_s": 34.439,
"avg_ms": 0.156, "max_ms": 570.836}, {"op": "MPI_Wait", "calls": 1327122, "total_s": 3.746, "avg_ms": 0.003, "ma...

[...]
```

The model (`[← llm]`) requests `memcpy_summary`, `gap_histogram`, `mpi_summary`; the harness (`[local]`) runs these tools and returns the responses in JSON format. Then the LLM updates its reasoning, and the next turn starts.

Output: ranked, grounded hypotheses

Each hypothesis carries a bottleneck type, its phase, an impact / confidence, an Amdahl speedup bound, and **Description** of the bottleneck found, numerical **Evidence**, and **Suggestion** for fixing it. Every claim should be tied back to profile numbers.

PerfAdvisor

Hypotheses — rank_683330_rocpd-0.db

memory_bound · phase: Kernel3D · impact: high · action: code · conf: high · runtime 61.0% speedup 44–156%

Description
Device-to-device memory copies consume 27.2s of wall time in the Kernel3D phase (61.0% of phase duration), transferring 566.8 GB across 155K copies. All copies are on stream 0 (the default/copy stream) while kernels run on stream 9. The effective bandwidth is ~20.8 GB/s, well below the MI250X's theoretical device memory bandwidth. The copies are in the 2-12 MB range, which should achieve higher throughput.

Evidence
D2D copies: 155,039 transfers, 27.2s total, 566.8 GB, effective 20.8 GB/s. All on stream_id 0. Dominant sizes: 3.98MB (103K copies, 17.0s) and 2.99MB (51K copies, 10.0s). 86% of sampled copies overlap with kernel execution, suggesting some pipelining already exists.

Suggestion
Investigate whether D2D copies can be replaced with in-place operations or reduced through better memory management (e.g., pointer swapping instead of copying). If copies are for halo exchange packing, consider using GPU-initiated communication (GPU-aware MPI with device pointers) to eliminate the intermediate copy step. Consider using multiple copy engines/streams to improve D2D bandwidth utilization.

PerfAdvisor typically returns 3 to 8 of these ranked hypothesis blocks per analyzed profile.

Validation: Synthetic benchmarks

Validation Benchmark

A synthetic benchmark suite—one for CUDA (`bench/cuda/`) and one for HIP (`bench/hip/`)—is used to validate/test PerfAdvisor.

Design rules

- 8 scenarios—each designed with one intentional bottleneck
- Opaque kernel names (`kernel_a`,...) to avoid leaking the answer to PerfAdvisor
- Only use pathologies that a systems-level profiler (e.g. Nsight Systems) can see—no pathologies that can only be diagnosed from hardware counters
- Bottleneck names and three expected/accepted fixes for each bottleneck are stored separately in a "ground-truth" JSON

#	Injected bottleneck	GPUs
01	Kernel-launch overhead (10k tiny launches)	1
02	CPU sync stall (`cudaStreamSynchronize` per launch)	1
03	PCIe transfer-bound (H2D → compute → D2H)	1
04	Transfer/compute overlap missing (serialized streams)	1
05	Unnecessary host staging, intra-node P2P	4
06	MPI load imbalance (ranks stall at `MPI_Barrier`)	4
07	Host-staged collective (`MPI_Allreduce`)	4
08	Inter-node halo exchange, host-staged	8

Benchmark Results

Model	Det@1	Det@3	Cov any/full
gemini-3.1-pro-preview *	.65±.28	.92±.20	89% / 16%
gemini-3.5-flash †	.62±.14	.68±.13	74% / 30%
claude-fable-5 ‡	.51±.10	.75±.07	90% / 46%
gpt-5.6-sol	.40±.09	.48±.09	81% / 32%
gpt-5.6-terra	.40±.05	.44±.07	74% / 42%
gpt-5-mini	.33±.06	.46±.20	76% / 28%
gemini-2.5-pro	.33±.06	.42±.10	67% / 28%
claude-sonnet-5	.31±.13	.52±.15	71% / 22%
claude-haiku-4-5	.29±.06	.52±.09	70% / 27%

Metrics

Detection — did a hypothesis name the injected *bottleneck*, and how highly ranked:

- **Det@1** — top-ranked hypothesis is correct (headline)
- **Det@3** — a correct hypothesis is in the top 3

Coverage — share of the ~3 expected *fixes* addressed (judge scores each 0/1/2):

- **Cov any** — % of expected fixes at least partly addressed (≥1)
- **Cov full** — % of expected fixes fully addressed (=2)

Asterisks

* gemini-3.1-pro: **19/48** — 29 lost to 429 (rate limits exceeded). Not comparable because the hardest cases weren't run.

† gemini-3.5-flash: **47/48** — 1 lost to a 503 (servers overloaded).

‡ claude-fable-5: **45/48** — 3 lost to Anthropic 529 (servers overloaded).

`--reasoning-effort` was set to `high` for the frontier models (fable-5, gpt-5.6-sol, gemini-3.1-pro), `medium` for the rest. `claude-haiku-4-5` and `gemini-2.5-pro` have no thinking control, so they ignore it.

Case studies

Test: MILC/QUDA lattice QCD with GDR on/off

Two Nsight Systems profiles of a six-solve CG campaign on a $48^3 \times 64$ lattice (8 MPI ranks, NERSC Perlmutter, A100). They differ only in `QUDA_ENABLE_GDR`. Without GPUDirect RDMA, every inter-node halo exchange is (unnecessarily) staged through host memory.

Running `analyze` on the *GDR-off* profile, PerfAdvisor (Claude Opus 4.6) finds "62.4 s of GPU idle time in 10–100 ms gaps" and traces it to serialized halo exchanges:

> *The GPU must wait while pack → D2H → MPI → H2D → unpack completes on the host before the next kernel can launch.*

	GDR off	GDR on
H2D + D2H transfers	355,778	< 4,000
Transfer wall-time	11.3 s	1.0 s
Total wall-time	baseline	**−43%**

PerfAdvisor's Recommendation:

> *Enable GPU-direct RDMA (GDR) to eliminate the host-staging D2H → MPI → H2D path.*

...which is precisely the correct fix.

Test: Surfacing hidden pathologies in an HMC run

A 4-GPU MILC/QUDA HMC run (2 RHMD trajectories) on NVIDIA B200 (Blackwell), analyzed in multi-rank mode. Ranks first disagreed on the phase count, but re-running with `--max-phases 3` recovered the expected (setup → RHMD trajectories → tear-down) structure. Running `perf-advisor analyze` (Opus 4.6) then surfaced two pathologies before I ever opened the GUI:

> *cpu_launch_overhead: The fermion force kernels are launched one at a time with large CPU-side delays between them... The CPU appears to spend ~1 second between launches on host-side computation...*

Hmm, did we miss a GPU-offload flag when compiling?

> *memory_bound: The D2D copies on stream 7 are serialized with gauge field loads...*

Back in the Nsight Systems GUI, both were exactly as described: host-side gaps before every multishift inversion (then an H2D copy before the GPU resumes), and large serialized D2D copies between Dslash calls where only overlapped P2P transfers should be. Time to optimize some code!

Takeaway: PerfAdvisor flags in minutes what might take hours to notice in the GUI—then you dig into it visually.

Wrap-up

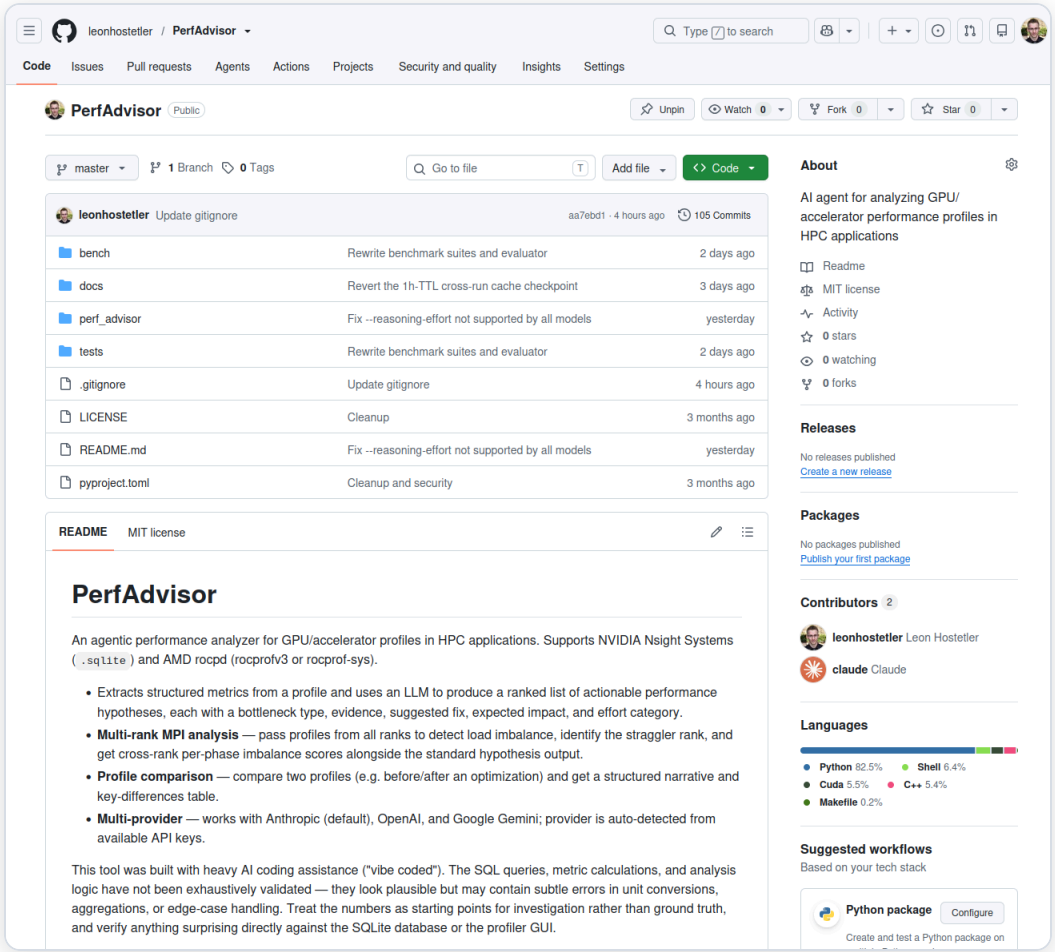
Summary

- PerfAdvisor is an agent that turns hours of manual profile forensics into an automated, evidence-grounded diagnosis.
- Works across AMD (ROCm Systems Profiler) and NVIDIA (Nsight Systems), with Anthropic / OpenAI / Gemini backends.

> ***Key takeaway:*** *PerfAdvisor quickly surfaces pathologies that would take you hours of digging to find on your own—solving the needle in a haystack problem*

Try it: `github.com/leonhostetler/PerfAdvisor`

Copyright: MIT License (open source, highly permissive)



Acknowledgments

- Thanks to **Steven Gottlieb**, **Xiangyu Jiang**, and **Peter Boyle** for useful conversations
- and to the **MILC Collaboration** for lattice configurations and compute time
- ROCm profiles were generated on Frontier (ALCC); NVIDIA profiles on Perlmutter (ERCAP)
- The author is supported in part by the U.S. DOE Office of Science (SciDAC: HEP, LAB 22-2580)
- Claude Code (Sonnet 4/5, Opus 4.6/4.8) provided significant assistance in the development of PerfAdvisor

Thank you!

APPENDIX

Feature deep-dives

Phase detection · multi-rank · prompt caching · prompt engineering · compare

Feature: phase detection

Why. Init, JIT compilation, and warmup look nothing like the steady-state solve. Without segmentation, one-off startup costs average into the hot loop and a transient masquerades as a bottleneck.

How it segments

The timeline is split into fixed-width bins. Each bin gets a *fingerprint*: its normalized per-kernel time distribution plus its idle fraction. Candidate boundaries are proposed where behavior changes:

- Jensen–Shannon divergence peaks between adjacent bin fingerprints[2]
- rocTX / NVTX range edges
- ends of MPI barrier clusters
- active ↔ idle transitions

How it chooses k (number of phases)

Dynamic programming finds the segmentation into k phases that minimizes total within-phase fingerprint divergence; k itself is chosen by a complexity penalty, so it doesn't over-segment noise into spurious phases.

Every downstream metric — GPU busy time, gap histograms, MPI wait—is then computed *per phase*, and every hypothesis carries its phase tag.

> *So JIT / init noise cannot masquerade as a steady-state bottleneck — the analysis code lives in* `analysis/phases.py`.

[2] J. Lin, "Divergence measures based on the Shannon entropy," *IEEE Trans. Inf. Theory*, 37(1):145–151, 1991.

Feature: multi-rank analysis

Problem. An MPI run emits one profile per rank. "The job is slow" hides *which* rank stalls, in *which* operation, in *which* phase — and ranks drift, so their phase boundaries don't line up in wall-clock time.

If multiple profiles are passed to PerfAdvisor, it will automatically attempt to run in multi-rank mode.

What it needs

1. Per-rank profiles with MPI instrumentation:
   - **Nsight Systems** — capture with `-t mpi`.
   - **rocprof-sys** — MPI intercepted.
   - **rocprofv3** — does *not* intercept MPI, so `capabilities.has_mpi = False` and cross-rank imbalance is unavailable.
2. Run PerfAdvisor with multiple profiles: `perf-advisor analyze /profiles/rank_*.db`

What it does

- Aligns phase boundaries across ranks so like is compared with like.
- Scores cross-rank imbalance per phase from GPU busy time + MPI wait time—the two ways a rank falls behind.
- Selects the straggler(s) and attributes the lag to specific operations.
- Pre-seeded as the `cross_rank_summary` tool so the agent starts multi-rank runs already knowing the imbalance picture.

Note: In multi-rank mode, the local analysis is performed across all provided profiles to obtain cross-rank metrics useful for identifying rank imbalances, etc. However, the agent loop will still only run on a single one of the profiles. I.e. the remote LLM still only studies (via local tools) a single profile, but it has knowledge of cross-rank metrics because of what was computed locally across all profiles and injected into its initial prompt.

Feature: prompt caching

Why. The agent loop is multi-turn, and each turn re-sends a large, mostly-unchanged context prefix—system prompt + tool catalog + pre-seeded summary + the growing tool-call history. Re-billing that prefix every turn is the dominant cost of a run.

Per-provider caching

- **Anthropic** — a sliding window of `cache_control` breakpoints: 1 permanent (system + tools + summary) + 2 floating (the two most recent turns) ⇒ ≈ 75–80% billable-input saving on a typical ~18-turn run.
- **OpenAI** — caches long prefixes automatically (~50%).
- **Gemini** — explicit context caching (~50–65%).

Capability-gated features degrade gracefully: sent optimistically, and on an unsupported-parameter rejection the loop warns once and retries without them (the `--reasoning-effort` fallback is the reference).

Cost is estimated and previewed

- A full `analyze` run is typically 15–60k input tokens + 1–3k output.
- Every `analyze` / `compare` prints a pre-flight token + cost estimate (with the cache breakdown) and asks to confirm *before* any paid call— `--yes` skips it in batch.
- Levers to reduce cost: Reduce `--max-phases` (largest single lever on context size), reduce `--max-turns`, or use a cheaper model.
- `summary` (Stage 1 local analysis) is free—run it first to decide whether the agent is even warranted.

Feature: prompt engineering

What goes into the prompt

The static system prompt (the cached prefix) is assembled from several blocks:

- **Role & goal** — "You are an expert GPU performance engineer..."; emit a ranked hypothesis list
- **Output schema** — the 8 `bottleneck_type` values, an `action_category` (fix-effort: `runtime_config` → `algorithm`), and a `confidence` level
- **Metric glossary** — exact definitions so fields aren't misread
- **SQL schema reference** — vendor-specific (CUPTI / rocpd) so `sql_query` calls are well-formed
- **Capability block** — which data is present/absent (*"no MPI data → don't raise MPI hypotheses"*)
- **Hardware context** — GPU, CU count, peak HBM BW, HBM, L2, clocks, thread/reg/shmem limits

The initial prompt *pre-seeds* some information (profile + phase summary + cross-rank summary) so the LLM starts off with at least that information.

Grounding constraint

A profile leaks enough (kernel names, sizes) for the LLM to *guess the application* and answer from prior knowledge—a hallucination risk. For example, it will sometimes invent non-existent build / environment flags. To fix this, we added a grounding constraint:

- Default: the prompt requires every suggestion be grounded in the profile data alone—don't infer the app's purpose or suggest options from memory.
- To disable this constraint, opt in with `--allow-app-knowledge`. Use only when a human will vet the output.

The LLM already knows quite a bit about QUDA and knows it's dealing with QUDA from the kernel names in the profile. With `--allow-app-knowledge`, the model sometimes names the precise and correct fix (e.g. set `QUDA_ENABLE_GDR=1`), but other times, it hallucinates a flag that does not exist (e.g. set `QUDA_ENABLE_CUDA_GRAPH=1`).

Feature: comparing two profiles

A `compare` mode is included for A/B testing: `perf-advisor compare A.sqlite B.sqlite`

How it works

- Both profiles are summarized independently (Stage 1), then a pre-computed structural diff is injected into a single LLM prompt—no tool-use loop, so one call.
- Output: a plain-English narrative plus a key-differences table ordered by magnitude of change.
- Three ordered modes are auto-selected: `phase_aware` (if phase detection finds same phases in both), `summary` (if the phases differ but kernel overlap ≥ 20%), `summary_no_kernel` (if kernel overlap < 20%).

When to use it

- Before / after an optimization—flip a flag, change a launch config, swap an algorithm, and quantify the effect.
- Regression checks across code versions or library updates.

> ***Limitation:*** *Cross-format comparison (rocpd vs nsys) is currently disallowed—the two formats' kernel-name schemas and metrics don't line up. Both profiles must be the same format; cross-format diffing may be added in the future.*

APPENDIX

Other details

Data privacy · local tools

Data privacy and risks

What leaves the machine

Running `analyze` or `compare` sends the extracted structured summary to the configured provider (Anthropic / OpenAI / Google)—each provider's retention policy then applies. This includes:

- **kernel names** — full demangled template names, which reveal algorithmic structure
- **NVTX / rocTX marker text** — arbitrary user strings (project names, identifiers)
- timing metrics, grid/block dims, transfer stats, MPI op names

The raw `.sqlite` is never transmitted — only the summary and follow-up tool results. This matters at sites with export-control or data-governance policies.

Analyze fully locally

- `perf-advisor summary` runs Stage 1 only—phase detection, per-phase metrics, kernel/MPI/memcpy tables—entirely on the machine. No LLM, no API key, nothing sent externally, and free. It is also the ground truth for checking any hypothesis the agent later makes.

Untrusted profiles

- Profile fields are inserted verbatim into the prompt, so a crafted profile could embed instruction-like text (prompt injection) or misleading names. The system prompt flags this data as untrusted, but only analyze profiles you trust.
- `sql_query` is read-only (`mode=ro`) and row-capped—the blast radius is the profile DB.

Local tools (callable by the LLM)

Tool	Function	Notes
`profile_summary`	Wall-clock, GPU utilization, tables	pre-seeded
`phase_summary`	Execution-phase segmentation	pre-seeded
`cross_rank_summary`	Per-rank GPU/MPI wait imbalance	pre-seeded (multi-rank)
`top_kernels`	Top kernels by time + launch overhead	
`gap_histogram`	Inter-kernel idle-gap distribution	
`memcpy_summary`	Transfer volume (H2D / D2H / xGMI)	
`mpi_summary`	MPI breakdown (rocprof-sys / Nsight)	
`marker_ranges`	rocTX / NVTX annotation ranges	
`stream_summary`	Per-stream GPU utilization	
`get_table_schema`	Column inspection	
`sql_query`	Arbitrary read-only SQL on the profile	**most used!**

APPENDIX

Benchmark details

Synthetic benchmarks

Benchmark Procedure

1. Run the 8 CUDA benchmark scenarios on Perlmutter (A100) and profile with Nsight Systems
2. Run PerfAdvisor:
    i. 6 times on each of the 8 profiles to get a measure of the variance in model answers
    ii. Repeat for 9 different models ranging from older (`claude-haiku-4-5`) to frontier (`claude-fable-5`)
3. For each of the 432 (8 profiles x 9 models x 6 repetitions) tests, PerfAdvisor returns a few sets of (suspected bottleneck + evidence + suggested fix)
4. A fixed judge model (`claude-opus-4-8`) scores each of the PerfAdvisor outputs against the ground-truth expectation (known injected bottleneck + accepted fixes)
    i. A *Detection@1* is counted if the *first* suggestion returned by PerfAdvisor matches the ground-truth bottleneck and the suggested fix names a mechanism specific to that scenario
    ii. A *Detection@3* is counted if the "correct" answer is in the top 3 suggestions returned by PerfAdvisor
    iii. *Coverage* grades the fixes, not the diagnosis: on runs where the bottleneck was detected, the judge scores each of the scenario's expected fixes on a 0 / 1 / 2 scale (0 = unaddressed, 1 = partially, 2 = fully addressed). Reported as two percentages of the expected fixes: Cov-any (scored ≥ 1) and Cov-full (scored exactly 2).

Synthetic benchmarks: A closer look (the fraction of Det@3 hits)

Over the six runs, did a given model find the injected bottleneck every time, sometimes, or never?

Model	01 launch	02 sync	03 transfer	04 overlap	05 p2p stage	06 mpi imbal	07 allreduce	08 halo
gemini-3.1-pro *	6/6	6/6	2/2	0/1	0/1	0/1	1/1	0/1
gemini-3.5-flash †	6/6	6/6	5/6	4/5	2/6	3/6	4/6	3/6
claude-fable-5 ‡	6/6	6/6	6/6	5/5	0/6	4/4	6/6	1/6
gpt-5.6-sol	5/6	5/6	6/6	0/6	0/6	1/6	6/6	0/6
gpt-5.6-terra	5/6	6/6	5/6	0/6	0/6	0/6	6/6	0/6
gpt-5-mini	6/6	0/6	5/6	4/6	0/6	4/6	6/6	1/6
gemini-2.5-pro	6/6	0/6	0/6	4/6	0/6	4/6	6/6	0/6
claude-sonnet-5	5/6	2/6	5/6	5/6	1/6	0/6	5/6	3/6
claude-haiku-4-5	6/6	6/6	5/6	1/6	0/6	4/6	6/6	0/6

Asterisks

* gemini-3.1-pro: **19/48** — 29 lost to **429** (rate limits exceeded). Not comparable because the hardest cases weren't run.

† gemini-3.5-flash: **47/48** — 1 lost to a **503** (servers overloaded).

‡ claude-fable-5: **45/48** — 3 lost to Anthropic **529** (servers overloaded).

Columns **05** and **08** are red down the field: two scenarios defeat almost everyone. This could indicate bad tests, e.g., bottlenecks not inferrable from the profile or presence of a more prominent (unintended) bottleneck.

Synthetic benchmarks summary

What it shows

- Two of the scenarios, intra-node host-staging (05) and inter-node halo (08), are missed almost every time by all models. This could imply poorly designed scenarios
- Model blind spots are real. E.g., the GPT family misses completely on transfer/compute overlap (04) whereas fable, sonnet, and flash do well on the scenario
- `claude-fable-5` is the most consistent model—scoring 6/6 on Det@3 across the easy/mid categories and with the best coverage (90% / 46%); only the two hard cases beat it.
- `claude-sonnet-5` performs surprisingly poorly on this benchmark

Caveats

- Benchmark scenarios were designed with significant LLM assistance (Claude Code)
- Benchmark scenarios could inadvertently introduce secondary/unintended bottlenecks
- A *miss* by a smart LLM could be due to a poorly designed scenario that the LLM sees through
- PerfAdvisor performance on the benchmarks is judged by LLM (`claude-opus-4-8`)
- The same Claude judge scores Claude/GPT/Gemini alike → may result in same-family bias
- Error bars record *precision*, not judge bias
- There is no bottleneck-free control profile

APPENDIX

More case studies

QUDA_ENABLE_P2P

Test: MILC/QUDA lattice QCD with `QUDA_ENABLE_P2P`

The same six-solve CG campaign, now with `QUDA_ENABLE_P2P=3` (enable QUDA's own optimized P2P) or `QUDA_ENABLE_P2P=0` (let MPI do its thing). A subtler, less visually obvious profile than the GDR case.

Running `perf-advisor compare` (Opus 4.6) leads with the right story: kernel time barely moves, so the 29% wall-clock win "comes entirely from reduced non-compute overhead." Running `perf-advisor analyze` on the *P2P-off* profile, PerfAdvisor pinpoints it: 348,208 small (~892 KB) P2P transfers at 40.9 GB/s on a non-overlapped stream—well under the A100's ~100 GB/s NVLink link.

	QUDA P2P off	QUDA P2P on
GPU kernel time	baseline	< 1% change
GPU idle time	baseline	**−62%**
Eff. P2P bandwidth	40.9 GB/s	62.7 GB/s
Total wall-time	baseline	**−29%**

PerfAdvisor's Recommendation:

> *Overlap halo transfers with interior computation by staging the boundary and interior dslash kernels on separate streams.*

Technically correct—though it targets the *communication pattern* rather than the QUDA flag.