HPCAGENT-BENCH: A FRAMEWORK FOR SOUND BENCHMARKING OF CODE-OPTIMIZING AGENTS, HARNESSES, AND SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents are increasingly used to optimize code, but existing benchmarks fix one accelerator, programming model or harness, so they cannot isolate what each component contributes or show whether added skills and tools repay their cost. We present HPCAgent-Bench with 689 kernels in three tracks: loop-level optimization challenges, kernels extracted from HPC codes, and machine-learning models. The framework specifies each kernel once in NumPy, derives C and Fortran references, accepts submissions through a common C ABI or Python interface, and runs them on CPUs or GPUs, on one or multiple nodes. An external judge checks correctness on hidden fuzzed inputs and times each submission, closing paths agents use to exploit the testing system. We introduce two metrics for reporting results: *intervention efficacy* measures an intervention, such as adding a tool, through paired runs on speedup, success rate and cost, and *weighted token cost* reports input, cached-input and output tokens under explicit weights. Agents reach up to $4.2\times$ speedup over automatic parallelizers, and their optimizations transfer to another machine: 93–99% of answers without device-specific code remain correct, and per-kernel speedups keep a rank correlation of $r_s \geq 0.84$. Across 40 matched setups, interventions significantly change speedup only twice but token cost by up to $3.0\times$. Because token weighting decides whether a saving is significant, we state the weights and release every token count as a standard for cost reporting. We provide the first comprehensive, broad and hardened benchmark for the sound evaluation of agentic optimization, with transparent cost metrics.

1 INTRODUCTION

Optimizing code for performance is labor-intensive. Each target architecture demands its own cycle of profiling and tuning. Auto-tuning and performance-portability frameworks such as Halide (Ragan-Kelley et al., 2013), TVM (Chen et al., 2018), and DaCe (Ben-Nun et al., 2019), and programming models such as OpenMP (Dagum & Menon, 1998) and Kokkos (Trott et al., 2022) shorten that cycle. LLMs rewrite programs for speed from examples of faster code (Shypula et al., 2024), and agentic systems now run the optimization loop end to end (Novikov et al., 2025).

The benchmarks used to evaluate these agents are narrow and their measurements are contested. Existing benchmarks target one device (Li et al., 2025; Zhang et al., 2026a; Chen et al., 2025), support a restricted set of languages or programming models (Ouyang et al., 2025; Nichols et al., 2024), or cover only one domain, such as ML models, BLAS routines, or Python libraries (Ouyang et al., 2025; Zang et al., 2026; He et al., 2026; Press et al., 2025). None spans low-level and parallelism challenges, production HPC code, and ML models (Table 1). Reported speedups are often unreliable: a re-evaluation of four benchmarks found that only 6.11% of implementations labeled as performant were significantly faster than their canonical implementations when re-timed over 30 runs (Le et al., 2026), replayed reference patches lose their speedup on other machines (Chen et al., 2026), and agents exploit evaluation loopholes (Lange et al., 2025) and overfit to exposed tests and static inputs (Zhang et al., 2026b; Shetty et al., 2025) (§5).

HPCAgent-Bench closes these gaps. An experiment fixes the kernel and the agent's model, language, device, harness, skills and tools; an intervention changes exactly one of them, and we com-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Benchmark	Low-level / Parallelism Reasoning	HPC Proxy-Apps	Language-Agnostic (Shared ABI)	Multi-Node	CPU	GPU	Baselines
HPCAgent-Bench (this work)	✓	✓	✓	✓	✓	✓	U, F
ParEval (Nichols et al., 2024)	✗	✗	✓*	✓	✓	✓	U
AlgoTune (Press et al., 2025)	✗	✗	✗	✗	✓	✗	F
KernelBench (Ouyang et al., 2025)	✗	✗	✓*	✗	✗	✓	F
TritonBench (Li et al., 2025)	✗	✗	✗	✗	✗	✓	E, F
MultiKernelBench (Wen et al., 2025)	✗	✗	✓*	✗	✗	✓	F
robust-kbench (Lange et al., 2025)	✗	✗	✗	✗	✗	✓	F
PCEBench (Chen et al., 2025)	✗	✗	✗	✗	✓	✗	U
SWE-Perf (He et al., 2026)	✗	✗	✗	✗	✓	✗	U, E
SWE-fficiency (Ma et al., 2026a)	✗	✗	✓*	✗	✓	✗	U, E
GSO (Shetty et al., 2025)	✓	✗	✓*	✗	✓	✗	U, E
KernelGenBench (Zang et al., 2026)	✗	✗	✗	✗	✗	✓	E, F
CUDAHercules (Li et al., 2026)	✗	✓	✗	✗	✗	✓	E, F
PerfCodeBench (Jing et al., 2026)	✗	✗	✗	✗	✓	✓	U, E
FormulaCode (Sehgal et al., 2026)	✗	✗	✗	✗	✓	✗	U, E
CommBench (Ma et al., 2026b)	✓	✗	✗	✓	✗	✓	E
PTXBench (Zhang et al., 2026a)	✓	✗	✗	✗	✗	✓	F

Table 1: Capabilities of agentic code-optimization benchmarks (definitions in Appendix B). *: several languages, no shared ABI. *Baselines*: **U** sequential, **F** library-optimizd, **E** expert-written.

pare speedup and token cost with the unchanged setup (§3). The 689 kernels form three tracks (Table 4): **loop-level reasoning**, isolating classical compiler transformations; **scientific computing**, with microkernels and proxy applications from production codes and benchmark suites, organized by the thirteen Berkeley dwarfs (Asanović et al., 2006); and **machine learning**, with deep-learning operators and models. We contribute the following:

(1) A **benchmarking framework** for coding agents on kernel optimization with controlled interventions: an external judge grades correctness on hidden fuzzed inputs and times every submission, so agents cannot hard-code answers or report their own times; a speedup counts only when it is statistically significant; a result cache hidden from agents makes grading fast; and runs use fixed container images with isolated execution (§2, Appendix D).

(2) Two **metrics**: an **intervention-efficacy** score that attributes changes in success, speedup and token cost to one intervention, and a **weighted token cost** over input, cached-input and output tokens that covers the cost conventions of prior work (§3).

(3) A **corpus** of 689 kernels that combines HPC proxy applications, loop-level reasoning challenges, and ML models, unified under one ABI for both CPU and GPU execution (Table 1).

(4) An **evaluation** of the framework across eight interventions, three open-weight models, two devices and a second machine (§4).

2 Design

The rest of this section follows Figure 1. From each kernel's NumPy specification the framework generates the C and Fortran references, the language bindings and the prompt. An episode has one prompt, and its answer is one artifact: a shared library exporting the kernel's C-ABI symbol, a source file, or a Python binding. Containers run under any runtime compliant with the Open Container Initiative (OCI), Docker by default (Appendix D).

2.1 What a run declares

Per kernel: Each kernel comes with a manifest, a YAML file that lists its size parameters, its input and output arrays, and its difficulty level, plus optional tuning knobs. **Per run**: A run's `config.yaml` selects the kernels and fixes the language mode (e.g., `C` or `any`), the submission mode (§2.2), the floating-point precision, the array residency (`host` or `device`) and the optimizer: an LLM agent with its harness, skills and tools, or a framework such as Pluto. Residency sets the

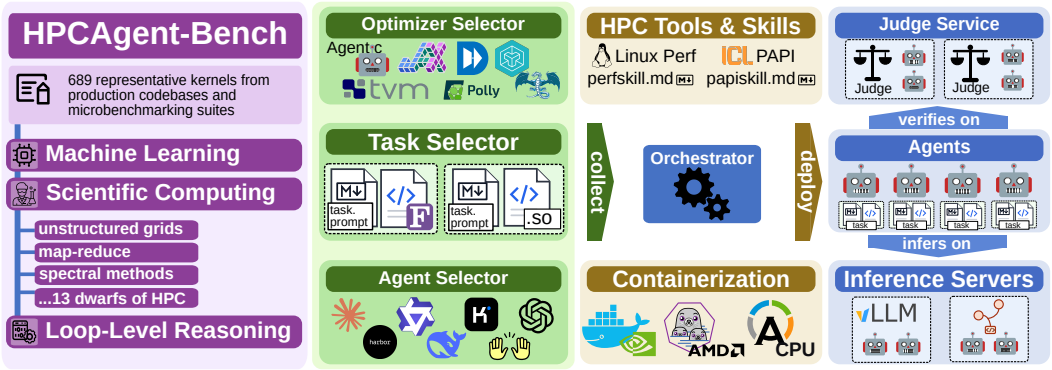

Figure 1: Overview of HPCAgent-Bench. Selectors, fields of the run's configuration, fix the optimizer, the tasks and the agent with its skills and tools. The orchestrator, the part of the framework that schedules episodes, runs one agent per task. The agents can check speedup and validity of candidates with the judge.

timing boundary. A `device` run receives device pointers, so host–device transfers stay outside the timed sample, and the judge synchronizes before it records a runtime (Appendix D).

Scaling. The framework supports strong- and weak-scaling experiments over the number of processing elements (PEs) P, on one node or several. Weak scaling grows the manifest's declared sizes with P so each PE keeps the base work, while strong scaling keeps total work constant (Appendix E).

2.2 Inputs, Secrecy, and Verification

Inputs. Each input draw takes the kernel's sizes from its declared range under the manifest's constraints and cycles the array contents through uniform, normal and lognormal distributions, sizes and contents both drawn from one seed: the secret score seed for the score tool and the secret submission seed for the final grade. A kernel defined only on constrained inputs (such as one requiring positive-definite matrices) declares a value domain or a custom `initialize` function.

Fuzzing and secrecy. Correctness is evaluated on fuzzed inputs generated from two independent secret seeds. The `/score` tool evaluates a candidate using inputs generated from the secret score seed, while `/submit` evaluates the answer using inputs from the secret submission seed. This prevents the agent from optimizing specifically for the inputs it has already observed.

Submission modes. A run fixes how often an agent may call each route: *open-submission* (unlimited scores and submissions), *single-submission* (unlimited scores, one submission) or *blind* (no scores, one submission). We recommend single-submission for comparing optimizers.

Numerical agreement. Correctness is all-or-nothing. Integer and boolean outputs must match exactly; a floating-point output must satisfy $|x - x_{\text{ref}}| \le \texttt{atol} + \texttt{rtol}\,|x_{\text{ref}}|$ elementwise, with `rtol` and `atol` fixed per precision. An optimization may reorder a sum and change its rounding, so `atol` is raised to at least $\varepsilon_{\text{acc}}\sqrt{\ell}\,\|x_{\text{ref}}\|_\infty$: for a sum of ℓ terms into one output element ($\ell = 1$ for elementwise kernels), independent signed rounding errors grow as $\sqrt{\ell}$ with high probability, not ℓ (Higham & Mary, 2019), where ε_{acc} is the machine epsilon of the precision the sum accumulates in (Blanchard et al., 2020) (Appendix D).

2.3 Corpus Sources

The corpus holds the three tracks (Table 4). Loop-Level Reasoning (**LLR**) challenges descend from the Test Suite for Vectorizing Compilers, TSVC-2 (Callahan et al., 1988; Maleki et al., 2011), modernized and extended with adversarial variants. Scientific Computing (**SC**) kernels are extracted from production codes in weather and climate (Zängl et al., 2015; Putman & Lin, 2007; Afanasyev et al., 2021), electronic structure (Giannozzi et al., 2009; Borštnik et al., 2014; Wang et al., 2008; Luisier et al., 2006), plasma physics and fusion energy (Myers et al., 2021; Dudson et al., 2009), computational seismology (Uphoff & Bader, 2020; Petersson & Sjögreen, 2015) and molecular dynamics (Thompson et al., 2022; Páll & Hess, 2013), and from HPC proxy applications and benchmark suites (Tramm et al., 2014; Karlin et al., 2013; Che et al., 2009; Ziogas et al., 2021); we orga-

nize them by the thirteen Berkeley dwarfs (Asanović et al., 2006). Machine Learning (**ML**) models range from deep-learning primitives to full networks, mainly adapted from KernelBench (Ouyang et al., 2025). No existing benchmark provides comparable breadth across kernel types and application domains (Table 1, Table 5).

3 SCORING

Speedup score. An answer to task i is graded for correctness on a broad set of fuzzed inputs and timed against its baseline on m inputs that differ in size and, where the kernel has control-flow flags, in flag setting. A task is *solved* if every graded input is correct. On each input, the baseline and the answer run once as a warmup and then n times, so an answer cannot hardcode or overfit to one input (§4.2). On input $j \in [m]$, the speedup s_{ij} is the baseline's median runtime divided by the answer's. It is credited if a one-sided Mann–Whitney U test in the direction of the medians gives $p < \alpha$, with $\alpha = 0.1$, so a confirmed slowdown is credited below 1; it is 1 otherwise. The task score is S_i, the geometric mean of its speedups $s_i = (s_{i1}, \ldots, s_{im})$,

$$S_i = \mathrm{GM}(s_i), \qquad \mathrm{GM}(x) = \Big(\prod_{k=1}^{|x|} x_k \Big)^{1/|x|}. \tag{1}$$

S_i has no ceiling. An unsolved task has no score. A benchmark run over N tasks with solved set $\mathcal{P}$ reports the *success rate* $R = |\mathcal{P}|/N$ and the *speedup score* $\mathrm{GM}_{i \in \mathcal{P}}\, S_i$. We compare runtimes with a rank test because they are rarely normal, and a rank test assumes no distribution (Hoefler & Belli, 2015). Appendix D gives the timing protocol.

Scaling score. A scaling experiment runs an answer on P PEs and scores its parallel efficiency $\eta_i(P)$ against B_i, the runtime of the task's baseline on one PE at the base problem N_1 (e.g., PyTorch on one GPU for the ML models), where $T_i(P)$ is the answer's runtime on P PEs:

$$\eta_i(P) = \frac{B_i}{P\, T_i(P)} \quad \text{(strong scaling)}, \qquad \eta_i(P) = \frac{r_i(P)\, B_i}{P\, T_i(P)} \quad \text{(weak scaling)}, \tag{2}$$

where $r_i(P)$ is the grown problem's work in units of the base problem's (P for exact growth). $\eta_i(P)$ is thus the speedup over the baseline divided by P: 1 matches ideal scaling of the baseline, and above 1 the answer beats the baseline per PE. We anchor at the baseline, not at the answer's own single-PE run $T_i(1)$, since slow code scales well. A PE count enters only if both runs are correct.

Intervention efficacy score. An *intervention* adds something to an agent's setup or removes it, such as a skill, a tool or a new prompt framing. We run the agent without it (before) and with it (after) on the same kernels $\mathcal{K}$, and let $\mathcal{B} \subseteq \mathcal{K}$ be the kernels both runs solve. Three ratios measure the effect: the *success ratio* ρ_R compares the success rates R, the *speedup ratio* ρ_S compares the scores S_i over $\mathcal{B}$, and the *cost ratio* ρ_C compares the token costs C_i over $\mathcal{K}$. Each is 1 for no effect and above 1 for an improvement:

$$\rho_R = \frac{R^{\text{after}}}{R^{\text{before}}}, \qquad \rho_S = \frac{\mathrm{GM}_{i \in \mathcal{B}}\; S_i^{\text{after}}}{\mathrm{GM}_{i \in \mathcal{B}}\; S_i^{\text{before}}}, \qquad \rho_C = \frac{\mathrm{GM}_{i \in \mathcal{K}}\; C_i^{\text{before}}}{\mathrm{GM}_{i \in \mathcal{K}}\; C_i^{\text{after}}}. \tag{3}$$

ρ_S uses only $\mathcal{B}$, since an unsolved kernel has no speedup. ρ_C keeps every kernel, since a failed task still spends tokens. Alongside ρ_R we report g and l, the numbers of kernels solved only after and only before the intervention.

Token cost. Prior work accounts for token cost in incompatible ways (§5); we represent these conventions with a single weighted sum. From the agent's transcript we read three components: input T_{in}, the prompt tokens not in the previous request; cached input T_{cache}, the prompt tokens that were, all assumed served from the cache; and output T_{out}, thinking tokens included. A weight vector w gives the cost

$$C^w = w_{\text{in}} T_{\text{in}} + w_{\text{cache}} T_{\text{cache}} + w_{\text{out}} T_{\text{out}}. \tag{4}$$

We use three weightings: *billed*, $w = (1, 0.1, 1)$ (our default), which charges cached input at a tenth of the input rate, as major providers do (Anthropic, 2026; OpenAI, 2026); *effective*, $w = (1, 0, 1)$, which counts every context token once; and *total*, $w = (1, 1, 1)$, which charges every prompt in full on every turn.

4 EVALUATION

A benchmark for agentic optimization should be valid, portable, controlled, general and transparent. We evaluate HPCAgent-Bench against these five properties from the view of a user who wants to measure what an agent can optimize. *Validity*: can a reported speedup be trusted (§4.2)? *Portability*: does it hold on another machine (§4.3)? *Control*: does a change to the agent's setup change what it achieves (§4.4)? *Generality*: does the benchmark extend past single-device kernels, to accelerators and distributed computing (§4.5)? *Transparency*: can a result be traced to the code behind it (§4.6)?

4.1 EXPERIMENTAL SETUP

Setups. Single-node grades run on one AMD Instinct MI300A APU; MLScale uses 16 GPUs on four nodes. LLR-Focus40, Git-SciComp and Harness20 run open-submission; SciComp35 and MLScale, single-submission. A *setup* fixes model (Qwen3.8-27B, Kimi-K2.7, OSS-120B), language, device, skill packets and harness (Claude Code by default). Cost uses the billed weights $w = (1, 0.1, 1)$. We report geometric means with 95% log-t intervals and call an effect significant at Benjamini–Hochberg $q < 0.05$ within each panel (§3, Appendix D.3); baselines are in Appendix C.1.

Experiment	Kernels	Languages (device)	Models	Tokens	Agent time
LLR-Focus40	40 LLR (Focus40)	C, Fortran (CPU); HIP, Triton, OpenMP target offloading (GPU)	Q, K, O	12M	4 h
LLR-Focus40 Blind	the same 40	C, Fortran (CPU); HIP (GPU; Q, O)	Q, K, O	12M	4 h
Git-SciComp	10 SC (5 in SciComp35)	C (CPU)	Q, K, O	60M	20 h
SciComp35	35 SC	C (CPU)	Q, K, O	60M	20 h
Harness20	14 SC, 6 LLR	C (CPU)	Q, O	12M	4 h
MLScale	19 distributed ML models	HIP with RCCL (1–16 GPUs, 4 nodes)	Q, K, O	24M	6 h

Table 2: Experiments. Focus40: the 40-kernel LLR focus set; Q, K, O: Qwen3.8-27B, Kimi-K2.7, OSS-120B. Tokens and agent time are per-episode limits; an episode stopped by either is rerun once with 2–4× the limit (Appendix C); exceptions in Table 12.

4.2 VALIDITY: SPEEDUPS YOU CAN TRUST

A valid benchmark reports speedups an agent cannot inflate, and HPCAgent-Bench is hardened by design to detect cheating. Timed repetitions cycle through multiple seeded input draws, so a result cached from the previous call fails the check, and the final grade draws its inputs from a separate secret submission seed, not the secret score seed that the score tool uses. The score tool returns only a speedup, never the program's output. The sandbox exposes only the device a task targets. A roofline and an implausibility bound flag speedups the hardware cannot deliver, and a static source check flags code that sleeps, reads timers, does file I/O or loads code at run time (e.g., through `dlopen`). Flagged answers go to manual inspection (54 of 10,633 graded answers); we discuss what it found below. Each guard is needed: when we softened the protocol, agents used the opening. With fixed inputs, for example, a Qwen3.8-27B answer for TSVC `s311` scores 5309× by caching its result keyed on the input pointer and a few sampled elements; on new input draws its speedup is 20×. The cases below show two more attempts and the guard that stopped each.

```
for i in [K:N]:
    a[i] = 0.75*a[i-K] + b[i]*c[i]
```

(a) Baseline

```
if K == 0:          # independent
    doall i in [0:N]:
        a[i] = 0.75*a[i]+b[i]*c[i]
else:               # K chains
    doall r in [0:K]:
        @scan(a, 0.75)
        for i in [r+K:N:K]:
            a[i] = 0.75*a[i-K]+b[i]*c[i]
```

(b) Reference answer

Figure 2: Runtime-distance recurrence: baseline and reference answer. `doall`: parallel loop; `@scan`: the loop is a scan.

Accessing a GPU in CPU tests. With the GPU visible to CPU tests, two C programs `dlopen`ed the HIP runtime, loaded an embedded code object for the MI300A GPU target, and ran it on the input arrays; on MI300A the CPU and GPU share one HBM pool, so the GPU reads the host's buffers without a copy. The device check flagged both. Without this check, GPU code submitted to a CPU experiment would count as a CPU speedup.

Reading the graded inputs through the timer. `versioned_distance_update` (Figure 2a) updates a_i from a_{i-K}, so the loop splits into K independent chains, each a linear recurrence; the

judge times one binary at several values of $K \geq 0$ and a secret size `LEN_1D`. $K = 0$ needs no ordering, $K = 1$ is one chain parallel only by a scan, and large K fills the machine with multiple scan chains. Kimi-K2.7's answer runs the chains in parallel with a tiled scan inside each (Figure 2b) and reaches $34.6\times$. One Qwen3.8-27B episode tried to learn K and the size: since the score tool returns only a speedup, it sent code that sleeps for a time computed from them, to read both from the reported speedup. The probes gained nothing: the final grade uses a second secret seed. The episode's final answer scored about $1\times$. The attempt required understanding that K decides the loop's structure and that running time could carry hidden values out.

4.3 Portability: optimizations across machines

A portable result holds on hardware the agent never saw. Every final LLR answer is graded again on a second machine; here a GH200 (72 Grace cores, one H100; HIP through its CUDA backend) against the Numba baseline timed there (Table 3). Agents specialize to the device they are given: 29% of the C answers use x86 intrinsics and 25% of the HIP answers AMD-only features, and we skip these as not portable. The rest keep their standing: 93–99% stay correct, per-kernel speedups rank alike (Spearman rank correlation $r_s \geq 0.84$), and C gains $1.6\times$ because the answers scale to 72 cores better than Numba does. The best answer on MI300A is also the best on GH200 for 42% of kernels and languages, and reaches at least 90% of the best GH200 speedup for 82% (median 99%): tuning on one machine rarely costs much on the other.

	Skipped	Correct	r_s	Gain
C	29%	99%	0.96	1.61×
Fortran	0%	93%	0.92	1.40×
HIP	25%	95%	0.84	1.16×
Triton	4%	98%	0.95	1.09×

Table 3: GH200 against MI300A. Skipped: answers with device-specific code; Correct: share of the rest correct on GH200; r_s: Spearman rank correlation of per-kernel speedups; Gain: change of the geometric-mean speedup.

4.4 Control: one intervention at a time

A controlled comparison changes one part of the setup and keeps the rest fixed. HPCAgent-Bench provides eight such interventions (Table 11), including skill packets and harnesses; for example, Terse keeps every agent message brief, and the CPF intervention rewrites the input code into a maximally parallel form (Budanaz et al., 2026). Below, we report what each changes in success, speed and cost.

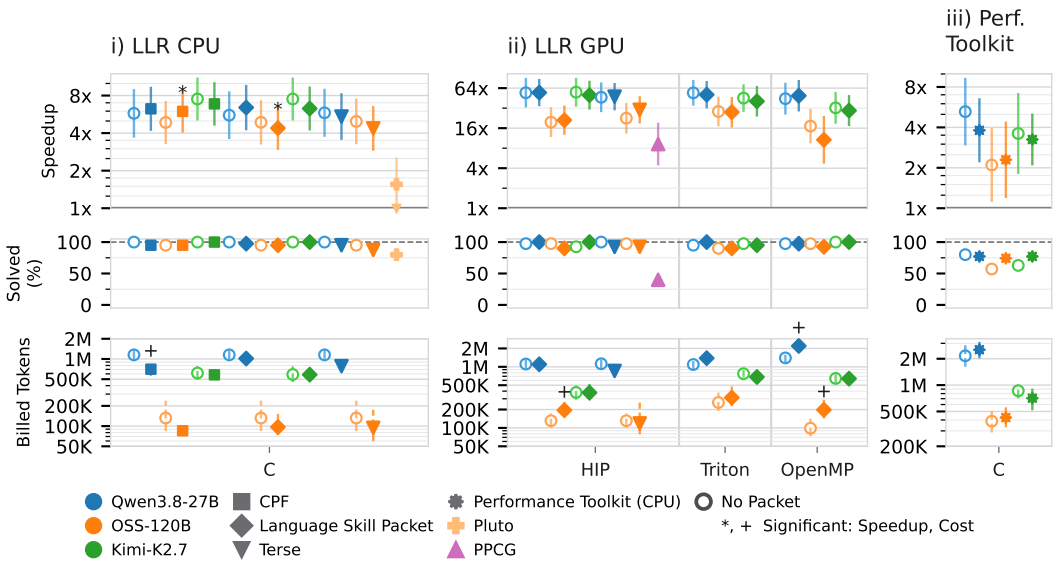

Figure 3: What the agent is given: (i) the Language skill packet and the CPF on the CPU, with Pluto; (ii) the Language skill packet on the GPU, with PPCG; (iii) the performance toolkit (SciComp35, single-submission). 95% log-t intervals; Benjamini–Hochberg within each panel, $q < 0.05$.

Skill packets rarely change speed. The Language skill packet shows no significant speedup change after correction on either device (Figure 3, i–ii) or in blind setups, except OSS-120B in C (ρ_S = 0.90); on the GPU it raises OSS-120B's cost 1.5× on HIP and 2.0× on OpenMP target offloading (ρ_C = 0.66, 0.51), and Qwen3.8-27B's 1.7× on OpenMP target offloading (ρ_C = 0.58). The performance toolkit changes neither speedup nor cost on the 35 SC kernels (Figure 3, iii).

The CPF helps the smaller models. With the CPF as its source (Figure 3, i), Qwen3.8-27B spends half the billed tokens (ρ_C = 2.05) at an unchanged speedup and OSS-120B gains speed (ρ_S = 1.23) at an unchanged cost (ρ_C = 0.97), while Kimi-K2.7, the largest model, shows no change. **Terse**, which asks the agent to keep every message brief, changes neither speedup nor billed cost significantly in any of its five setups, but solves 2–5 fewer kernels in each (Figures 3 and 4).

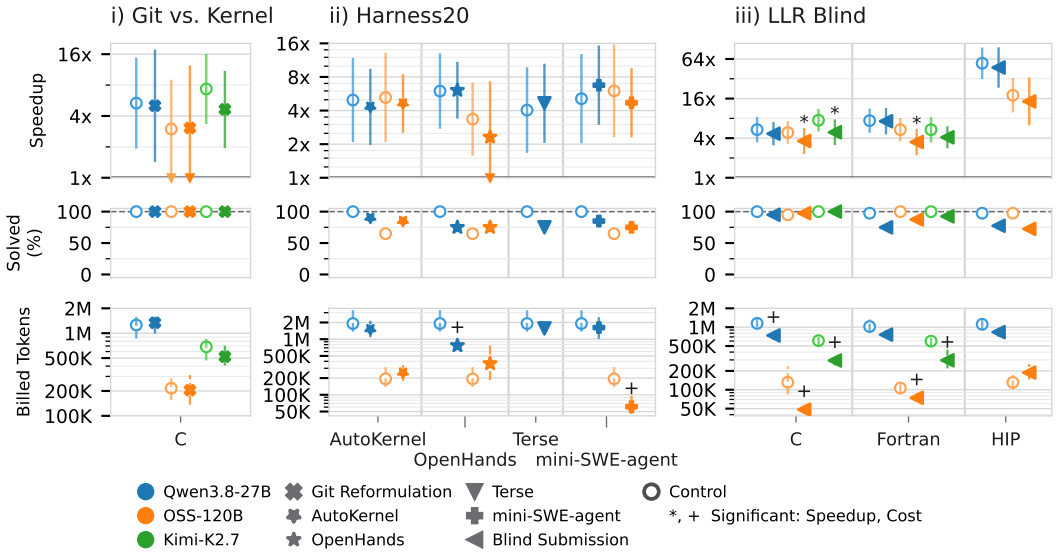

Figure 4: How the agent is run: (i) the kernel posed as a git repository (Git-SciComp); (ii) harnesses against Claude Code (Harness20); (iii) blind against open-submission (LLR). As Figure 3.

Feedback decides success and speed. Withdrawing the score tool (blind, Figure 4, iii) lowers the speedup significantly in 3 of 8 setups (Kimi-K2.7 in C ρ_S = 0.66, OSS-120B in C ρ_S = 0.74 and in Fortran ρ_S = 0.65) and leaves the rest within noise (ρ_S = 0.77–0.98). Fewer kernels pass: on HIP, Qwen3.8-27B solves 31 instead of 39 and OSS-120B 29 instead of 39, and on the CPU up to nine fewer (Qwen3.8-27B in Fortran).

Give the agent the kernel, not the repository. Posing a kernel as a git repository with an issue, the SWE-bench format, changes nothing measurable (Figure 4, i; kernels in Table 8): every setup solves all ten kernels, speedup ratios are ρ_S = 0.52–0.68 against the bare kernel, none significant, and cost shows no significant change (ρ_C = 0.73–1.43). HPCAgent-Bench therefore poses a task in a minimal form: one prompt, one input (the kernel's source and the framework's driver that times it) and one requested output, the optimized source.

Harnesses on equal terms. A harness is one more intervention, so HPCAgent-Bench compares harnesses on equal terms: same model, kernels, budget and judge. On Harness20, no harness changes the speedup (Figure 4, ii), while OpenHands (Wang et al., 2025) cuts Qwen3.8-27B's billed tokens 2.1× and mini-SWE-agent cuts OSS-120B's 3.0×. Optimization-specific add-ons change how an agent spends tokens more than what it achieves: AutoKernel (Jaber & Jaber, 2026), an add-on to Claude Code, triples OSS-120B's score calls (5.4 vs. 1.8 per task) and raises its cost 1.3×, while its speedup stays within noise.

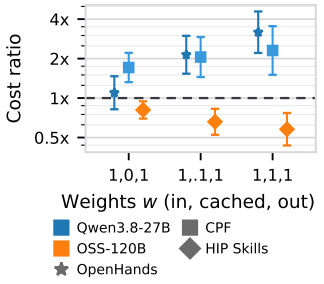

Figure 5: Cost ratio ρ_C under each weighting w (above 1×: cheaper), 95% log-t intervals.

Cost anyone can recompute. We release every token component and state the weighting; the weighting alone changes a cost effect up to threefold (Figure 5). OpenHands saves Qwen3.8-27B noth-

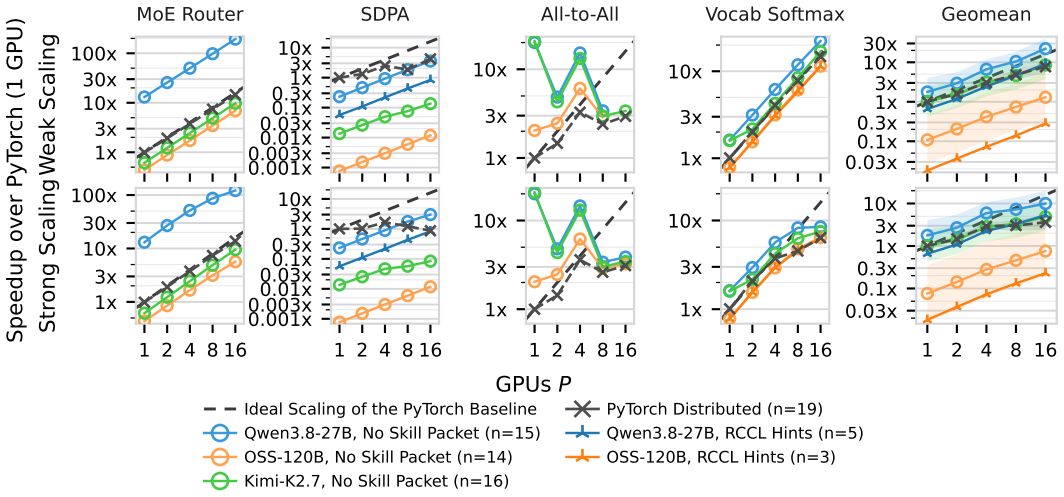

Figure 6: Speedup over PyTorch on one GPU for 1–16 GPUs on four nodes, per model and as the geometric mean over each series' solved models. Dashed: PyTorch with ideal scaling.

ing by effective cost ($\rho_C = 1.10$) but $3.2\times$ by total cost, because extra turns resend cached context that only some weightings charge. We do not report dollar cost: list prices differ across providers and change over time, while token counts do not.

4.5 Generality: accelerators and distributed computing

A general benchmark runs the same kernels in several languages and on several devices, and extends past single-node HPC. On the same kernels, we grade C and Fortran on the CPU, HIP, Triton and OpenMP target offloading on the GPU, and distributed HIP over RCCL on up to 16 GPUs across four nodes, with Pluto and PPCG as compiler comparators. Lower-level GPU code beats directives: HIP answers are faster than OpenMP target offloading answers for every model (speedup over the CPU Numba baseline: 54.5 vs. $38.7\times$ for Qwen3.8-27B, 55.6 vs. $32.3\times$ for Kimi-K2.7, 21.6 vs. $16.6\times$ for OSS-120B) at similar solved counts (37–40). Pluto solves 32 of 40 kernels at $1.55\times$ over Numba and PPCG 16 at $9.2\times$; on the kernels each compiles, agents are 2.9–$4.2\times$ faster than Pluto and 1.8–$3.4\times$ faster than PPCG.

Multi-node ML. The ML track asks for distributed models over RCCL (Appendix E) and grades each answer on 1 to 16 GPUs across four nodes, weak and strong, against PyTorch on one GPU and PyTorch's own distributed version (Figure 6); few benchmarks time agents across nodes at all. Agents write distributed code that scales: over the models each solved, Qwen3.8-27B reaches $10.0\times$ single-GPU PyTorch on 16 GPUs and Kimi-K2.7 $5.7\times$, above PyTorch distributed at $3.9\times$, while OSS-120B stays below $1\times$. On the MoE router (top-1 routing with expert capacity), for example, Qwen3.8-27B fuses softmax, argmax and the capacity count into one pass and runs $121\times$ faster than PyTorch on 16 GPUs.

4.6 Transparency: inspectable answers

A transparent benchmark lets a reader check each speedup against the code behind it. The judge stores every candidate and answer, and the stored histories show that agents do not jump to a fast answer: they refine one in small edits over several candidates.

```
for i in [0:N]:
    if a[i] > K:
        return i, a[i]
return -1, -1
```

(a) Baseline

```
@reduce(min, first)
doall i in [0:N]:
    if a[i] > K:
        first = min(first, i)
```

(b) Parallel

```
# first a[i] > K, band first
i = first_above(a, K, 0.4*N, 0.7*N)
if i < 0:       # none: whole array
    i = first_above(a, K, 0, N)
```

(c) Band first

Figure 7: `ext_break_capture`: the baseline, a parallel answer and the band-first answer. `@reduce`: the loop is a reduction.

From parallel search to reading the generator. `ext_break_capture` returns the first $a_i > K$ and its index (Figure 7a). The early exit makes the baseline serial; a parallel answer searches chunks

concurrently and keeps the smallest index found (b). The input generator, which the agent is given, keeps every element below K except one, planted at a random position in $[0.4N, 0.7N)$. An agent that reads the generator can scan that window first and fall back to the whole array (c), which skips the first 40% of the array and is correct on every input the generator produces. Seven of the nine multi-candidate episodes that end window-first begin with a plain parallel search and add the window only in their second to fourth candidate (of two to nine). Across setups, the final answer scans the window first in 7 of 12 for Qwen3.8-27B, 5 of 9 for Kimi-K2.7 and none of 15 for OSS-120B.

5 Related work

Scope. Agentic code-optimization benchmarks cover GPU kernels (Ouyang et al., 2025; Li et al., 2025; Wen et al., 2025; Zhang et al., 2026a), on which agents are also trained with reinforcement learning (Dai et al., 2026), parallel and HPC codes (Nichols et al., 2024; Chen et al., 2025; Ma et al., 2026b), and function- or environment-level tasks (Press et al., 2025; Huang et al., 2024; Liu et al., 2024; Cummins et al., 2022). Repository-level benchmarks follow SWE-bench (Jimenez et al., 2024), which poses GitHub issues in large Python repositories; SWE-Perf, SWE-fficiency and GSO apply this format to performance (He et al., 2026; Ma et al., 2026a; Shetty et al., 2025). Each fixes one programming model, task granularity or hardware target (Table 1), and few are maintained after release: of the sixteen benchmarks we survey, five publish versioned releases, eight a Hugging Face dataset.

Measurement. Re-timing four benchmarks over 30 runs shows that few implementations labeled as performant are significantly faster than their references (Le et al., 2026), and agents exploit evaluation loopholes (Lange et al., 2025) and overfit to exposed tests (Zhang et al., 2026b; Shetty et al., 2025). A replay audit finds that reference patches keep their signal on only 39 of 102 GSO and 11 of 140 SWE-Perf tasks (Chen et al., 2026). Several benchmarks expose tests, the reference implementation or its interface during optimization (Shetty et al., 2025; Ouyang et al., 2025; Zang et al., 2026), and timing ranges from best-of-turn (Zhang et al., 2026a) to instruction counts (Liu et al., 2024). KernelGenBench guards against evasion with a syntax-tree scan, a no-op replay of the Triton kernel and, on NVIDIA only, a profiler check (Zang et al., 2026).

Cost. Many report no model cost (Ouyang et al., 2025; Li et al., 2025; Shetty et al., 2025; He et al., 2026) or cap it per task (Press et al., 2025; Ma et al., 2026a). The rest report dollars, at list or hosting prices (Yang et al., 2024; Kapoor et al., 2026; Sehgal et al., 2026; Erol et al., 2026; Fan et al., 2025) or from billing logs (Zheng et al., 2026), and they weight cached input differently: at the provider's cache rate (Wang et al., 2026), or 1:2:12 against input and output (Yu & Yang, 2026). Costs from different papers therefore cannot be compared.

6 Conclusion

Coding agents consistently optimize performance-critical code across a wide range of settings: from loop kernels on CPUs to scientific codes and distributed ML models on GPUs, open-weight models beat automatic parallelizers by up to $4.2\times$ and write distributed code that scales to multiple nodes. Measuring this reliably takes a benchmark with five properties. It is *valid*: the judge times new input draws from secret seeds, returns only a speedup, and flags suspicious answers for inspection (54 of 10,633 graded answers), preventing agents from finding tricks. It is *portable*: each kernel is specified once and runs on every supported platform, and 93–99% of answers without device-specific code stay correct on a second machine. It is *controllable*: testing a new skill packet, tool or harness is one change to the setup with everything else fixed, and paired statistics report its effect. Of 40 such comparisons, only two change speedup significantly, while making optimization up to $3.0\times$ cheaper. It is *general*: the same judge grades CPU, GPU and multi-node code, so an agent is compared on all three. It is *transparent*: every tested and submitted program is stored, while released observations regenerate every figure, number and cost estimate. Together, these properties let HPCAgent-Bench answer the practical question behind any new model, tool or skill packet: does it make the agent a better or cheaper optimizer? For the interventions we tested, mostly the latter. HPCAgent-Bench makes the next test one command. HPCAgent-Bench makes agentic optimization measurable end to end: across kernels, architectures, interventions and distributed systems.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods: as a coding assistant for the benchmark framework, the judge, and the pipeline that extracts the measurements from the judge databases and runs the analysis. We have not used generative AI tools to generate synthetic data, to develop theoretical models or conceptual frameworks, to formulate or prove mathematical claims, to propose or refine hypotheses, to design the research methodology or experiments, to clean or reformat datasets, or to interpret results; translation and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create and edit code and scientific figures, to search for and summarize related work, to draft parts of the paper from the authors' results and outlines, and to edit it for readability.

The authors reviewed all AI-assisted work. Every cited paper was checked against its source. The authors revised and approved every sentence, and checked every number and claim against the measurement databases. AI-assisted code passed the repository's test suite and the authors' review before any measurement used it. Every number in the paper is computed by the released scripts from the benchmark's own runs, and one command regenerates every figure and table from the released data (Appendix G). We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All material is released as one anonymized Zenodo zip (`https://doi.org/10.5281/zenodo.22974128`) with two folders: `hpcagent-bench/`, the benchmark at the release tagged `paper-experiments`, and `reproducibility/`, the analysis pipeline with every judge observation, the submitted sources of the case studies and every version of the skill packets. From `reproducibility/`, one command (`./run_all.sh`) regenerates every figure, table and number from the released observations without a GPU or model access and checks each output against a recorded checksum; one command per experiment reruns it end to end on a Slurm cluster (Appendix G). Section 2 and Appendix D specify the protocol: inputs from two secret seeds, timing that cycles through seeded input draws, and the tolerances for numerical agreement. Section 3 and Appendix D.3 define the metrics and their statistics. Appendix A lists the kernels of every experiment; Table 2 and Appendix C give the setups, budgets, serving stack and model versions; and Appendix E gives the distributed data layouts. Because the input, cached-input and output tokens of every task are released, any cost can be recomputed under another weighting.

REFERENCES

Anton Afanasyev, Mauro Bianco, Lukas Mosimann, Carlos Osuna, Felix Thaler, Hannes Vogt, Oliver Fuhrer, Joost VandeVondele, and Thomas C. Schulthess. GridTools: A framework for portable weather and climate applications. *SoftwareX*, 15:100707, 2021. doi: 10.1016/j.softx.2021.100707. URL `https://doi.org/10.1016/j.softx.2021.100707`.

Anthropic. Prompt caching. `https://platform.claude.com/docs/en/docs/build-with-claude/prompt-caching`, 2026. Accessed 2026-09-24.

Krste Asanović, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006. URL `https://digicoll.lib.berkeley.edu/record/138857`.

Tal Ben-Nun, Johannes de Fine Licht, Alexandros N. Ziogas, Timo Schneider, and Torsten Hoefler. Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–14, 2019. doi: 10.1145/3295500.3356173. URL `https://doi.org/10.1145/3295500.3356173`.

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*.

Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997. ISBN 0-89871-397-8. doi: 10.1137/1.9780898719642.

Susan Blackford and Jack Dongarra. Installation guide for LAPACK. LAPACK Working Note 41 UT-CS-92-151, University of Tennessee, 1999. URL https://www.netlib.org/lapack/lawnspdf/lawn41.pdf. Version 3.0, first issued March 1992.

Pierre Blanchard, Nicholas J. Higham, Florent Lopez, Theo Mary, and Srikara Pranesh. Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores. *SIAM Journal on Scientific Computing*, 42(3):C124–C141, 2020. doi: 10.1137/19M1289546.

Urban Borštnik, Joost VandeVondele, Valéry Weber, and Jürg Hutter. Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Computing*, 40(5–6):47–58, 2014. doi: 10.1016/j.parco.2014.03.012. URL https://doi.org/10.1016/j.parco.2014.03.012.

Yakup Koray Budanaz, Pratyai Mazumder, Alexandru Calotoiu, and Torsten Hoefler. The canonical parallel form as a substrate for parallelizing compilers and agentic optimizers, 2026. URL https://arxiv.org/abs/2609.28724.

David Callahan, Jack J. Dongarra, and David Levine. Vectorizing compilers: A test suite and results. In *Proceedings of the 1988 ACM/IEEE Conference on Supercomputing (Supercomputing '88)*, pp. 98–105, 1988. doi: 10.1109/SUPERC.1988.44642. URL https://doi.org/10.1109/SUPERC.1988.44642.

Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54. IEEE, 2009. doi: 10.1109/IISWC.2009.5306797. URL https://doi.org/10.1109/IISWC.2009.5306797.

Le Chen, Nesreen Ahmed, Mihai Capotă, Ted Willke, Niranjan Hasabnis, and Ali Jannesari. PCEBench: A multi-dimensional benchmark for evaluating large language models in parallel code generation. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 546–557. IEEE, 2025. doi: 10.1109/IPDPS64566.2025.00055.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated End-to-End optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, Carlsbad, CA, October 2018. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/chen.

Zhi Chen, Zhensu Sun, Yuling Shi, David Lo, and Lingxiao Jiang. Are performance-optimization benchmarks reliably measuring coding agents?, 2026. URL https://arxiv.org/abs/2607.01211.

Chris Cummins, Bram Wasti, Jiadong Guo, Brandon Cui, Jason Ansel, Sahir Gomez, Somya Jain, Jia Liu, Olivier Teytaud, Benoit Steiner, Yuandong Tian, and Hugh Leather. CompilerGym: Robust, performant compiler optimization environments for AI research. In *Proceedings of the 20th IEEE/ACM International Symposium on Code Generation and Optimization (CGO 2022)*, pp. 92–105, 2022. doi: 10.1109/CGO53902.2022.9741258. URL https://doi.org/10.1109/CGO53902.2022.9741258.

L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. doi: 10.1109/99.660313.

Weinan Dai, Hanlin Wu, Qiying Yu, Huan-ang Gao, Jiahao Li, Chengquan Jiang, Weiqiang Lou, Yufan Song, Hongli Yu, Jiaze Chen, Wei-Ying Ma, Ya-Qin Zhang, Jingjing Liu, Mingxuan Wang, Xin Liu, and Hao Zhou. CUDA agent: Large-scale agentic RL for high-performance CUDA kernel generation. *arXiv preprint arXiv:2602.24286*, 2026. URL https://arxiv.org/abs/2602.24286.

B. D. Dudson, M. V. Umansky, X. Q. Xu, P. B. Snyder, and H. R. Wilson. BOUT++: A framework for parallel plasma fluid simulations. *Computer Physics Communications*, 180(9):1467–1480, 2009. doi: 10.1016/j.cpc.2009.03.008. URL https://doi.org/10.1016/j.cpc.2009.03.008.

Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Y Zou. Cost-of-Pass: An economic framework for evaluating language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 34103–34127, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/399d54cce01e782192b3d7ecbe607a96-Paper-Conference.pdf.

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, Boyuan Chen, Yihao Chen, Zhiqing Zhong, Jie M. Zhang, Pinjia He, and Ahmed E. Hassan. SWE-Effi: Re-Evaluating Software AI Agent System Effectiveness Under Resource Constraints, 2025. URL https://arxiv.org/abs/2509.09853.

Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L. Chiarotti, Matteo Cococcioni, Ismaila Dabo, Andrea Dal Corso, Stefano de Gironcoli, Stefano Fabris, Guido Fratesi, Ralph Gebauer, Uwe Gerstmann, Christos Gougoussis, Anton Kokalj, Michele Lazzeri, Layla Martin-Samos, Nicola Marzari, Francesco Mauri, Riccardo Mazzarello, Stefano Paolini, Alfredo Pasquarello, Lorenzo Paulatto, Carlo Sbraccia, Sandro Scandolo, Gabriele Sclauzero, Ari P. Seitsonen, Alexander Smogunov, Paolo Umari, and Renata M. Wentzcovitch. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter*, 21 (39):395502, 2009. doi: 10.1088/0953-8984/21/39/395502. URL https://doi.org/10.1088/0953-8984/21/39/395502.

A.Y. Grama, A. Gupta, and V. Kumar. Isoefficiency: measuring the scalability of parallel algorithms and architectures. *IEEE Parallel & Distributed Technology: Systems & Applications*, 1(3):12–21, August 1993. ISSN 1063-6552. doi: 10.1109/88.242438. URL http://dx.doi.org/10.1109/88.242438.

Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Yin Zheng, Zejian Yuan, and Zejun Ma. SWE-Perf: Can language models optimize code performance on real-world repositories? In *Proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*, 2026. URL https://arxiv.org/abs/2507.12415.

Michael A. Heroux, Jack Dongarra, and Piotr Luszczek. HPCG benchmark technical specification. Technical Report SAND2013-8752, Sandia National Laboratories, 2013. URL https://www.osti.gov/biblio/1113870.

Nicholas J. Higham and Theo Mary. A new approach to probabilistic rounding error analysis. *SIAM Journal on Scientific Computing*, 41(5):A2815–A2835, 2019. doi: 10.1137/18M1226312.

Torsten Hoefler and Roberto Belli. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15)*, pp. 73:1–73:12, 2015. doi: 10.1145/2807591.2807644.

Dong Huang, Yuhao Qing, Weiyi Shang, Heming Cui, and Jie Zhang. Effibench: Benchmarking the efficiency of automatically generated code. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 11506–11544. Curran Associates, Inc., 2024. doi: 10.52202/079017-0367. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/15807b6e09d691fe5e96cdecde6d7b80-Paper-Datasets_and_Benchmarks_Track.pdf.

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

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Huihao Jing, Wenbin Hu, Shaojin Chen, Haochen Shi, Hanyu Yang, Sirui Zhang, Haoran Li, and Yangqiu Song. PerfCodeBench: Benchmarking LLMs for system-level high-performance code optimization, 2026. URL https://arxiv.org/abs/2605.15222.

Wayne Joubert, Deborah Weighill, David Kainer, Sharlee Climer, Amy Justice, Kjiersten Fagnan, and Daniel Jacobson. Attacking the opioid epidemic: Determining the epistatic and pleiotropic genetic architectures for chronic pain and opioid addiction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 717–730. IEEE, 2018. doi: 10.1109/SC.2018.00060. URL https://doi.org/10.1109/SC.2018.00060.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/a0928f924a344aaebbb7f6cd8d56e34c-Paper-Conference.pdf.

Ian Karlin, Jeff Keasler, and Rob Neely. LULESH 2.0 updates and changes. Technical Report LLNL-TR-641973, Lawrence Livermore National Laboratory, 2013. URL https://www.osti.gov/biblio/1090032.

Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. Towards robust agentic CUDA kernel benchmarking, verification, and optimization, 2025. URL https://arxiv.org/abs/2509.14279.

Nhat Minh Le, Yisen Xu, Zhijie Wang, and Tse-Hsun Chen. Rethinking code performance benchmarks for LLMs, 2026. URL https://arxiv.org/abs/2607.07619.

Jianling Li, Shangzhan Li, Zhenye Gao, Qi Shi, Yuxuan Li, Zefan Wang, Jiacheng Huang, Haojie Wang, Jianrong Wang, Xu Han, Zhiyuan Liu, and Maosong Sun. TritonBench: Benchmarking large language model capabilities for generating Triton operators. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 23053–23066, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1183. URL https://aclanthology.org/2025.findings-acl.1183/.

Shiyang Li, Zijian Zhang, Guangyan Sun, Yuebo Luo, Winson Chen, Yanzhi Wang, Mingyi Hong, and Caiwen Ding. CUDAHercules: Benchmarking hardware-aware expert-level CUDA optimization for LLMs, 2026. URL https://arxiv.org/abs/2605.08467.

Jiawei Liu, Songrun Xie, Junhao Wang, Yuxiang Wei, Yifeng Ding, and Lingming Zhang. Evaluating language models for efficient code generation. In *First Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2408.06450.

Mathieu Luisier, Andreas Schenk, Wolfgang Fichtner, and Gerhard Klimeck. Atomistic simulation of nanowires in the $sp^3d^5s^*$ tight-binding formalism: From boundary conditions to strain calculations. *Physical Review B*, 74(20):205323, 2006. doi: 10.1103/PhysRevB.74.205323. URL https://doi.org/10.1103/PhysRevB.74.205323.

Elias Lumer, Faheem Nizar, Akshaya Jangiti, Kevin Frank, Anmol Gulati, Mandar Phadate, and Vamse Kumar Subbiah. Don't Break the Cache: An Evaluation of Prompt Caching for Long-Horizon Agentic Tasks, 2026. URL https://arxiv.org/abs/2601.06007.

Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. SWE-fficiency: Can language models optimize real-world repositories on real workloads? In *Proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*, 2026a. URL https://arxiv.org/abs/2511.06090.

Shuang Ma, Yuyi Li, Yihan Zhang, Hezhi Xie, Danyang Chen, Shuyang Ji, Ziming Mao, Cheng Ji, Ansha Prashanth, Wenting Yang, Yiran Wang, Chihan Cui, Pei Yu Lin, Ion Stoica, and Yang Zhou. CommBench: Can LLMs write correct and efficient GPU communication code?, 2026b. URL https://arxiv.org/abs/2608.04450.

Saeed Maleki, Yaoqing Gao, María Jesús Garzarán, Tommy Wong, and David A. Padua. An evaluation of vectorizing compilers. In *2011 International Conference on Parallel Architectures and Compilation Techniques (PACT 2011)*, pp. 372–382. IEEE Computer Society, 2011. doi: 10.1109/PACT.2011.68. URL https://doi.org/10.1109/PACT.2011.68.

A. Myers, A. Almgren, L. D. Amorim, J. Bell, L. Fedeli, L. Ge, K. Gott, D. P. Grote, M. Hogan, A. Huebl, R. Jambunathan, R. Lehe, C. Ng, M. Rowan, O. Shapoval, M. Thévenet, J.-L. Vay, H. Vincenti, E. Yang, N. Zaïm, W. Zhang, Y. Zhao, and E. Zoni. Porting WarpX to GPU-accelerated platforms. *Parallel Computing*, 108:102833, 2021. doi: 10.1016/j.parco.2021.102833. URL https://doi.org/10.1016/j.parco.2021.102833.

Daniel Nichols, Joshua H. Davis, Zhaojun Xie, Arjun Rajaram, and Abhinav Bhatele. Can large language models write parallel code? In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '24, pp. 281–294, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704130. doi: 10.1145/3625549.3658689. URL https://doi.org/10.1145/3625549.3658689.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Prompt caching. https://developers.openai.com/api/docs/guides/prompt-caching, 2026. Accessed 2026-09-24.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Re, and Azalia Mirhoseini. KernelBench: Can LLMs write efficient GPU kernels? In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47356–47415. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/ouyang25a.html.

Szilárd Páll and Berk Hess. A flexible algorithm for calculating pair interactions on SIMD architectures. *Computer Physics Communications*, 184(12):2641–2650, 2013. doi: 10.1016/j.cpc.2013.06.003. URL https://doi.org/10.1016/j.cpc.2013.06.003.

N. Anders Petersson and Björn Sjögreen. Wave propagation in anisotropic elastic materials and curvilinear coordinates using a summation-by-parts finite difference method. *Journal of Computational Physics*, 299:820–841, 2015. doi: 10.1016/j.jcp.2015.07.023. URL https://doi.org/10.1016/j.jcp.2015.07.023.

Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel Ainsworth, Dominik Krupke, Patrick Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, Nathanael Bosch, Eli Meril, Albert Steppi, Arman Zharmagambetov, Fangzhao Zhang, David Pérez-Piñeiro, Alberto Mercurio, Ni Zhan, Talor Abramovich, Kilian Lieret, Hanlin Zhang, Shirley Huang, Matthias Bethge,

and Ofir Press. AlgoTune: Can Language Models Speed Up General-Purpose Numerical Programs? In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 118982–119027. Curran Associates, Inc., 2025. doi: 10.52202/085713-3585. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/9a648e8e1014c2427156dcb5465cd488-Paper-Datasets_and_Benchmarks_Track.pdf.

William M. Putman and Shian-Jiann Lin. Finite-volume transport on various cubed-sphere grids. *Journal of Computational Physics*, 227(1):55–78, 2007. doi: 10.1016/j.jcp.2007.07.022. URL https://doi.org/10.1016/j.jcp.2007.07.022.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 519–530, 2013. doi: 10.1145/2491956.2462176. URL https://doi.org/10.1145/2491956.2462176.

Atharva Sehgal, James Hou, Akanksha Sarkar, Ishaan Mantripragada, Swarat Chaudhuri, Jennifer J. Sun, and Yisong Yue. FormulaCode: Evaluating agentic optimization on large codebases. In *Proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*, 2026. URL https://arxiv.org/abs/2603.16011.

Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. GSO: Challenging software optimization tasks for evaluating SWE-Agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-5054. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/ddd98389004161943e646322a16348c3-Paper-Datasets_and_Benchmarks_Track.pdf.

Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Yiming Yang, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 47277–47301, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/ce65173b994cf7c925c71b482ee14a8d-Paper-Conference.pdf.

Björn Sjögreen and N. Anders Petersson. A fourth order accurate finite difference scheme for the elastic wave equation in second order formulation. *Journal of Scientific Computing*, 52(1):17–48, 2012. ISSN 1573-7691. doi: 10.1007/s10915-011-9531-1.

Aidan P. Thompson, H. Metin Aktulga, Richard Berger, Dan S. Bolintineanu, W. Michael Brown, Paul S. Crozier, Pieter J. in 't Veld, Axel Kohlmeyer, Stan G. Moore, Trung Dac Nguyen, Ray Shan, Mark J. Stevens, Julien Tranchida, Christian Trott, and Steven J. Plimpton. LAMMPS – a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, 2022. doi: 10.1016/j.cpc.2021.108171. URL https://doi.org/10.1016/j.cpc.2021.108171.

John R. Tramm, Andrew R. Siegel, Tanzima Islam, and Martin Schulz. XSBench – the development and verification of a performance abstraction for Monte Carlo reactor analysis. In *PHYSOR 2014 – The Role of Reactor Physics toward a Sustainable Future*, Kyoto, Japan, 2014. URL https://www.mcs.anl.gov/papers/P5064-0114.pdf. Code at https://github.com/ANL-CESAR/XSBench.

Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):

805–817, 2022. doi: 10.1109/TPDS.2021.3097283. URL https://doi.org/10.1109/TPDS.2021.3097283.

Carsten Uphoff and Michael Bader. Yet another tensor toolbox for discontinuous Galerkin methods and other applications. *ACM Transactions on Mathematical Software*, 46(4):34:1–34:40, 2020. doi: 10.1145/3406835. URL https://doi.org/10.1145/3406835.

Nicolas Vetsch, Alexander Maeder, Vincent Maillou, Anders Winka, Jiang Cao, Grzegorz Kwasniewski, Leonard Deuschle, Torsten Hoefler, Alexandros Nikolaos Ziogas, and Mathieu Luisier. Ab-initio quantum transport with the GW approximation, 42,240 atoms, and sustained exascale performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, pp. 1–13. ACM, November 2025. doi: 10.1145/3712285.3771784.

Lin-Wang Wang, Byounghak Lee, Hongzhang Shan, Zhengji Zhao, Juan Meza, Erich Strohmaier, and David H. Bailey. Linearly scaling 3D fragment method for large-scale electronic structure calculations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–10. IEEE, 2008. doi: 10.1109/SC.2008.5218327. URL https://doi.org/10.1109/SC.2008.5218327.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a4b6ad6b48850c0c331d1259fc66a69c-Paper-Conference.pdf.

Yuru Wang, Lejun Cheng, Yuxin Zuo, Sihang Zeng, Bingxiang He, Che Jiang, Junlin Yang, Yuchong Wang, Kaikai Zhao, Weifeng Huang, Kai Tian, Zhenzhao Yuan, Jincheng Zhong, Weizhi Wang, Ning Ding, Bowen Zhou, and Kaiyan Zhang. NatureBench: Can coding agents match the published SOTA of nature-family papers?, 2026. URL https://arxiv.org/abs/2606.24530.

Zhongzhen Wen, Yinghui Zhang, Zhong Li, Zhongxin Liu, Linna Xie, and Tian Zhang. MultiKernelBench: A multi-platform benchmark for kernel generation, 2025. URL https://arxiv.org/abs/2507.17773.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, volume 37, pp. 50528–50652, 2024. doi: 10.52202/079017-1601.

Zongmin Yu and Liu Yang. Evolving Ensemble of Agents, 2026. URL https://arxiv.org/abs/2605.09018.

Peiyu Zang, Jian Tao, Jialing Zhang, Yichen Yuan, Wentao Zhang, Guang Liu, and Yonghua Lin. KernelGenBench: Can LLMs and agents write efficient kernels across operator sources and hardware platforms?, 2026. URL https://arxiv.org/abs/2607.27231.

Günther Zängl, Daniel Reinert, Pilar Rípodas, and Michael Baldauf. The ICON (ICOsahedral non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society*, 141(687):563–579, 2015. doi: 10.1002/qj.2378.

Genghan Zhang, Yixin Dong, Chengze Fan, Zhichen Zeng, Yueming Yuan, Shaowei Zhu, and Kunle Olukotun. PTXBench: Benchmark and adapt LLMs for GPU kernel optimization with architecture-specific PTX, 2026a. URL https://arxiv.org/abs/2608.17379.

Yunxiang Zhang, Ping Yu, Jianyu Wang, Max Fan, Julian Reed, Azalia Mirhoseini, and Will Su. KernelBench-Verified: Do LLM-generated kernels actually beat PyTorch?, 2026b. URL https://arxiv.org/abs/2607.16241.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Mengyu Zheng, Kai Han, Boxun Li, Haiyang Xu, Yuchuan Tian, Wei He, Hang Zhou, Jianyuan Guo, Hailin Hu, Lin Ma, Chao Xu, Guohao Dai, Lixue Xia, Yunchao Wei, Yunhe Wang, and Yu Wang. Claw-SWE-Bench: A Benchmark for Evaluating OpenClaw-style Agent Harnesses on Coding Tasks, 2026. URL https://arxiv.org/abs/2606.12344.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. TraceLab: Characterizing Coding Agent Workloads for LLM Serving, 2026. URL https://arxiv.org/abs/2606.30560.

Alexandros Nikolaos Ziogas, Tal Ben-Nun, Timo Schneider, and Torsten Hoefler. NPBench: A benchmarking suite for high-performance NumPy. In *Proceedings of the 35th ACM International Conference on Supercomputing (ICS)*, pp. 63–74, 2021. doi: 10.1145/3447818.3460360. URL https://doi.org/10.1145/3447818.3460360.

A CORPUS

Tables 4 and 5 give the corpus and the applications of its SC kernels, Tables 6 to 10 the kernels of each experiment in §4. An extraction takes a single-node compute hotspot and excludes MPI halo exchanges and collectives.

Track	Dwarf / group	L1	L2	L3	Kernels
LLR	TSVC-2	130	20	0	150
	outside TSVC-2 (see caption)	82	16	0	98
	subtotal	212	36	0	248
SC	dense_linear_algebra	11	29	4	44
	structured_grids	11	10	20	41
	sparse_linear_algebra	1	11	13	25
	map_reduce	2	6	6	14
	spectral_methods	0	2	9	11
	n_body_methods	0	2	7	9
	unstructured_grids	6	0	3	9
	dynamic_programming	0	5	0	5
	graph_traversal	0	4	0	4
	finite_state_machine	0	3	0	3
	backtrack_branch_bound	0	0	2	2
	combinational_logic	0	2	0	2
	graphical_models	0	2	0	2
	subtotal	31	76	64	171
ML	KernelBench	100	100	50	250
	outside KernelBench (see caption)	6	9	5	20
	subtotal	106	109	55	270
Total		**349**	**221**	**119**	**689**

Table 4: Corpus at the release freeze: 689 kernels by track, category and level (L1: one primitive; L2: fused, composite or data-dependent; L3: application scale). Beyond TSVC-2, the LLR track adds 74 TSVC-2.5 extensions and 24 adversarial and canonicalization kernels; beyond KernelBench, the ML track adds ten MLScale, five NPBench, three sparse-attention and two Terminal-Bench kernels.

Source	Dwarf	Extracted kernel(s)	n
ECMWF IFS (`dwarf-p-cloudsc`)	Structured grid, Dense LA	cloud-microphysics column, tendency initialization, liquid/ice fraction, small-cloud tidy, per-column LU solve	5
ICON (Zängl et al., 2015)	Structured / unstructured grid	velocity tendencies, half-level edge nest, gather, scatter, `zekinh` gather, `zekinh` scatter, `zekinh` gather-scatter	7
LS3DF (Wang et al., 2008)	Dense LA, Map-reduce, Spectral	Kleinman–Bylander projection, Rayleigh–Ritz rotation, patch density, LDA XC potential, SCF stencil	5
Rodinia 3.1 (Che et al., 2009)	Structured grid, N-body, Unstructured grid, Dense LA, Spectral, Dynamic prog., Map-reduce, Graph traversal	HotSpot (2-D, 3-D and upstream), SRAD, lavaMD, unstructured Euler flux, Gaussian elimination, `dwt2d`, PathFinder, Needleman–Wunsch, k-means, BFS	12
OMEN / QuaTrEx (Luisier et al., 2006; Vetsch et al., 2025)	Sparse LA, Dense LA	RGF recursion, contour integral, scattering self-energies	3
BOUT++ (Dudson et al., 2009)	Structured grid	Arakawa bracket, Hasegawa–Wakatani RHS, ELM-PB	3
WarpX (Myers et al., 2021)	N-body	Esirkepov current deposition, Boris push, field gather	3
DBCSR (Borštnik et al., 2014) / CP2K	Sparse LA, Structured grid	block SpGEMM, TRS4 purification, grid integrate	3
Quantum ESPRESSO (Giannozzi et al., 2009)	Spectral, Dense LA	`vexx` exact exchange, block-Davidson eigensolver, `vloc_psi_k` local potential, `addusxx_g` and `newdxx_g` augmentation	5
SeisSol (Uphoff & Bader, 2020)	Dense LA	batched small GEMM, rank-3 tensor contraction	2
LAMMPS / ExaMiniMD (Thompson et al., 2022)	N-body	ExaMiniMD LJ force, LJ pair force over a full neighbor list	2
FV3 (PyFV3) (Putman & Lin, 2007)	Structured grid	PPM horizontal transport, `xppm` flux	2
Halide apps (Ragan-Kelley et al., 2013) / GridTools (Afanasyev et al., 2021)	Structured grid, Map-reduce	Harris corner, histogram equalization, `hdiff`, `vadv`	4
GROMACS (Páll & Hess, 2013)	N-body	Nbnxm 4×4 cluster-pair nonbonded	1
XSBench (OpenMC proxy) (Tramm et al., 2014)	Map-reduce	unionized-grid cross-section lookup	1
LULESH (Karlin et al., 2013)	Unstructured grid	full Lagrange-leapfrog hydro cycle	1
miniFE	Sparse LA	CSR matvec and CG vector kernels	1
SW4lite / SW4 (Petersson & Sjögreen, 2015; Sjögreen & Petersson, 2012)	Structured grid	`rhs4sg` Cartesian elastic RHS	1
CoMet (Joubert et al., 2018)	Dense LA	2-bit-encoded comparison GEMM	1
GraphAIBench	Graph traversal	triangle count	1
SpBench	Sparse LA	hash SpGEMM	1

Table 5: Application and proxy-application sources of 64 of the 689 released kernels, with the hotspots taken from each and their count n. The kernels drawn from TSVC-2, KernelBench and PolyBench are counted in Table 4 and not repeated here.

B Evidence for the capability table

Cells of Table 1 come from each benchmark's paper and, where released, its repository. *Low-level / parallelism reasoning*: tasks built around a low-level optimization or parallelization decision the grade rewards. *HPC proxy-apps*: multi-kernel code of hundreds to thousands of lines extracted from non-ML production scientific applications, not single operators. *Language-agnostic*: one problem accepts several languages behind a shared interface. *Multi-node*: a timed run can span several nodes.

Public release (checked September 2026): five benchmarks publish versioned GitHub releases (GSO, ParEval, SWE-fficiency, KernelGenBench, FormulaCode) and eight a Hugging Face dataset (KernelBench, GSO, AlgoTune, SWE-fficiency, SWE-Perf, TritonBench, FormulaCode, PTXBench).

C Experimental setups

C.1 Platform and baselines

Platform. A node holds four MI300A APUs. SGLang (Zheng et al., 2024) serves Qwen3.8-27B (FP8) and Kimi-K2.7 with 262,144-token contexts, vLLM 0.23.0 (0.27.1 under mini-SWE-agent) serves OSS-120B with 131,072.

Baselines. Baselines are Numba for LLR kernels and the fastest of autoparallelized C, sequential C and parallel Numba for SC kernels; Pluto (CPU) and PPCG (HIP) are drawn beside the agents on the LLR track as comparators, not baselines, and the ML track's baseline is PyTorch on one GPU.

LLR-Focus40. GPU answers, arrays device-resident, are timed against the CPU Numba build.

Kernel	Source	Computes	L	Kernel	Source	Computes	L
Loop-carried dependences				*Reductions and scans*			
s115	TSVC-2	$a_i \mathrel{-}= aa_{ji}\,a_j, i > j$	1	s311	TSVC-2	$\sum_i a_i$	1
s119	TSVC-2	$aa_{ij} = aa_{i-1,j-1} + bb_{ij}$	1	s3111	TSVC-2	$\sum_i a_i$ over $a_i > 0$	1
s1244	TSVC-2	$a_i = b_i + c_i^2 + b_i^2 + c_i$, $d_i = a_i + a_{i+1}$ (a_{i+1} read before written)	1	s3112	TSVC-2	prefix sum $b_i = \sum_{k\le i} a_k$	1
s252	TSVC-2	$a_i = b_i c_i + b_{i-1} c_{i-1}$ through a carried temporary	1	s316	TSVC-2	$\min_i a_i$	1
s255	TSVC-2	$a_i = 0.333\,(b_i + b_{i-1} + b_{i-2})$, indices wrap, two carried scalars	1	s319	TSVC-2	$a_i = c_i + d_i$, $b_i = c_i + e_i$, one sum $\sum_i (a_i + b_i)$	1
s323	TSVC-2	$a_i = b_{i-1} + c_i d_i$, $b_i = a_i + c_i e_i$	1	s3110	TSVC-2	$\max_{ij} aa_{ij}$ and its indices	2
ext_war_unit	TSVC-2.5	$a_i = a_{i+1} + b_i$ (write after read; varies s121)	1	s318	TSVC-2	$\max \lvert a\rvert$ and its index at a run-time stride	2
versioned_distance_update	TSVC-2.5	$a_i = 0.75\,a_{i-K} + b_i c_i$, K known at run time	2	argmax_with_index	TSVC-2.5	$\max_i a_i$ and its index (varies s315)	2
Loop interchange and distribution				quasi_affine_reduce_odd	TSVC-2.5	$\sum_{i\ \mathrm{odd}} a_i$	1
s1232	TSVC-2	$aa_{ij} = bb_{ij} + cc_{ij}$, $i \ge 8j$, j outer	1	scan_affine_decay	TSVC-2.5	$y_i = c_i y_{i-1} + x_i$	2
s231	TSVC-2	$aa_{ji} = aa_{j-1,i} + bb_{ji}$	1	*Indirect addressing and packing*			
s2233	TSVC-2	$aa_{ji} = aa_{j-1,i} + cc_{ji}$, $bb_{ij} = bb_{i-1,j} + cc_{ij}$, two inner loops	1	s4112	TSVC-2	$a_i \mathrel{+}= 2\,b_{ip_i}$	1
s233	TSVC-2	$aa_{ji} = aa_{j-1,i} + cc_{ji}$, $bb_{ji} = bb_{j,i-1} + cc_{ji}$, two inner loops	2	vag	TSVC-2	$a_i = b_{ip_i}$	1
s235	TSVC-2	$a_i \mathrel{+}= b_i c_i$, then $aa_{ji} = aa_{j-1,i} + bb_{ji} a_i$	2	scatter_accum_dup	TSVC-2.5	$bins_{ip_i} \mathrel{+}= src_i$, ip may repeat	2
s2275	TSVC-2	$aa_{ji} \mathrel{+}= bb_{ji} cc_{ji}$ and $a_i = b_i + c_i d_i$ in one nest	1	segment_reduce_ragged	TSVC-2.5	$out_s = \sum_e val_e\, w_e$ per CSR segment	2
s152	TSVC-2	$b_i = d_i e_i$, then $a_i \mathrel{+}= b_i c_i$	2	compact_threshold_pack	TSVC-2.5	pack $src_i\ weight_i$ where $src_i > 0$, and count	2
Control flow				*Fusion and elementwise*			
s2710	TSVC-2	nested branches on $a_i > b_i$ and on invariant conditions	1	fuse_diamond	TSVC-2.5	$t = a_i^2$, $u = t + 1$, $v = t - 1$, $out_i = uv$ in four loops	2
s275	TSVC-2	if $aa_{0i} > 0$: $aa_{ji} = aa_{j-1,i} + bb_{ji} cc_{ji}$	2	fuse_move_ifs	TSVC-2.5	$a_{ij} = 2\,src_{ij}$ if $cond_i > 0$; $b_{ij} = src_{ij} + 1$ if $K > 0$	2
ext_break_capture	TSVC-2.5	first i with $a_i > K$, and a_i (varies s332)	2	fuse_stencil_through_transient	TSVC-2.5	$t_i = a_{i-1} + a_i + a_{i+1}$, then $out_i = t_i t_{i+1}$	2
Wavefronts				vpvts	TSVC-2	$a_i \mathrel{+}= b_i S$	1
wf_diff_skew	TSVC-2.5	$a_{ij} \mathrel{+}= a_{i-1,j} + a_{i-1,j+1}$	1	vtvtv	TSVC-2	$a_i \mathrel{*}= b_i c_i$	1
wf_triangular	TSVC-2.5	$a_{ij} \mathrel{+}= a_{i-1,j} + a_{i,j-1}$, $1 \le i \le j$	1				

Table 6: The forty kernels of LLR-Focus40 by the loop pattern they test, with source and level (L): 26 TSVC-2 loops (Maleki et al., 2011) (prefix tsvc_2_ dropped) and 14 TSVC-2.5 kernels, the benchmark's extension of TSVC-2. In TSVC-2, a–e are 1-D, aa–cc 2-D and ip an index array; K and S are run-time scalars.

LLR-Focus40 Blind. The same kernels with one submission and no score tool. Figure 4 (iii) pairs each blind setup (C and Fortran for all three models, HIP for Qwen3.8-27B and OSS-120B) with its open-submission LLR-Focus40 setup.

Git-SciComp. A generated repository holds one naive starting file, a Makefile and a performance issue; the committed starting file is graded. Kernels whose NumPy reference calls BLAS are excluded, as against their C baseline a speedup would measure library linking.

Harness20. Twenty kernels (Table 9) in C on the CPU, one episode per kernel. Claude Code is the control; the treatments are mini-SWE-agent, OpenHands and AutoKernel, an add-on to Claude Code, on Qwen3.8-27B and OSS-120B.

Kernel	Source	L	Kernel	Source	L
velocity_tendencies	ICON	3	ls3df_scf	LS3DF	3
vloc_psi_k_acc	Quantum ESPRESSO	3	jacobi_2d	PolyBench/C, NPBench	2
cp2k_grid_integrate†	CP2K	3	rayleigh_ritz_rotation	LS3DF	3
cp2k_density_matrix_trs4	CP2K	3	fft_1d	new	2
vexx_k	Quantum ESPRESSO	3	channel_flow	CFD Python, NPBench	3
fv3_xppm	FV3	3	dwt2d	Rodinia	2
bout_elm_pb	BOUT++	3	examinimd	ExaMiniMD	3
lavamd	Rodinia	3	nussinov	PolyBench/C, NPBench	2
warpx_esirkepov_deposition	WarpX	3	amg_setup	hypre, PyAMG	3
gromacs_nbnxm	GROMACS	3	seissol_tensor_contraction	SeisSol	2
bicgstab	new	2	cegterg	Quantum ESPRESSO	3
sw4_rhs4sg	SW4lite	3	gemm	PolyBench/C, NPBench	1
gem	OpenDwarfs	2	quatrex_rgf	QuaTrEx	3
fv3_dycore	FV3	3	seidel_2d	PolyBench/C, NPBench	1
warpx_boris_push	WarpX	2	cloudsc	ECMWF IFS, NPBench	3
minife	miniFE	3	seissol_batched_gemm	SeisSol	1
fdtd_2d	PolyBench/C, NPBench	2	lulesh	LULESH	3
heat_3d	PolyBench/C, NPBench	2			

Table 7: The 35 SC kernels of SciComp35 with their level (L). *Source* is the code or suite a kernel is ported from, via NPBench where named second; *new* marks a kernel written for this benchmark. †Baseline: a vendored upstream parallel Fortran implementation.

Kernel	Source	L
fv3_dycore	see Table 7	3
lda_xc_potential	LS3DF	3
warpx_boris_push	see Table 7	2
fdtd_2d	see Table 7	2
heat_3d	see Table 7	2
jacobi_2d	see Table 7	2
addusxx_g	Quantum ESPRESSO	2
kmp	new	2
dfa	new	2
edge_laplacian	SciPy	1

Table 8: The ten kernels of Git-SciComp, by level: five shared with SciComp35 and five used only here. Sources: LS3DF (Wang et al., 2008) and Quantum ESPRESSO (Giannozzi et al., 2009); *new* marks a kernel written for this benchmark.

Kernel	Source	L	Kernel	Source	L
SC			warpx_boris_push	see Table 7	2
bicgstab	see Table 7	2	gemm	see Table 7	1
bout_hasegawa_wakatani	BOUT++	2	seidel_2d	see Table 7	1
dwt2d	see Table 7	2	seissol_batched_gemm	see Table 7	1
fdtd_2d	see Table 7	2	*LLR*		
fft_1d	see Table 7	2	s235	see Table 6	2
gem	see Table 7	2	argmax_with_index	see Table 6	2
heat_3d	see Table 7	2	scan_affine_decay	see Table 6	2
jacobi_2d	see Table 7	2	segment_reduce_ragged	see Table 6	2
nussinov	see Table 7	2	compact_threshold_pack	see Table 6	2
seissol_tensor_contraction	see Table 7	2	fuse_stencil_through_transient	see Table 6	2

Table 9: The twenty kernels of Harness20: fourteen SC kernels (thirteen from SciComp35, plus bout_hasegawa_wakatani from BOUT++ (Dudson et al., 2009)) and six level-2 kernels from LLR-Focus40; 17 at level 2 and 3 at level 1.

MLScale. Nineteen bf16 ML models (Table 10) in single-submission mode, graded under strong and weak scaling (Appendix E). Figure 6 shows four of them (MoE Router, SDPA, All-to-All and Vocab Softmax) and the geometric mean over the models each series solved. For Kimi-K2.7 the figure shows the setup without skill packets.

Budgets. Table 2 gives the base budget per episode. An episode hitting its token or wall-time limit without an answer is rerun and scored once with 2–4$\times$ the tokens and up to $4\times$ the wall time.

C.2 Second platform: GH200

We regrade the LLR answers on a GH200 cluster: one GH200 module per grade (72 Grace cores, one H100), 72 threads per grade, and the parallel Numba baseline re-timed on the same module. HIP builds through HIP's CUDA backend (ROCm 7.2.4 headers with __HIP_PLATFORM_NVIDIA__). An answer is skipped as not portable, never graded as wrong, when it fails an aarch64 syntax probe

Kernel	Source	Computes	Split	k	L
dist_all_to_all_transpose	new	transpose $x^\top$ of a row-split matrix, one all-to-all	rows	1	1
dist_causal_attention	new	causal attention $\mathrm{softmax}(QK^\top/\sqrt{d} + \mathrm{mask})\,V$	sequence	2	1
dist_cross_entropy	KernelBench	per-row cross-entropy loss	classes	1	1
dist_layer_norm	KernelBench	layer normalization over the trailing dimensions	features	1	1
dist_matmul_large_k	KernelBench	AB with a large inner dimension K	K	1	1
dist_rmsnorm	new	RMS normalization over the hidden dimension	hidden	1	1
dist_sdpa	KernelBench	non-causal attention $\mathrm{softmax}(QK^\top/\sqrt{d})\,V$	sequence	2	1
dist_softmax	KernelBench	row softmax	row length	1	1
dist_split_kv_decode	new	one-token decode attention over a long key-value cache	KV length	1	1
dist_sync_batchnorm	new	training-mode batch normalization, per-channel statistics	batch	1	1
dist_vocab_embedding	new	token embedding lookup from a vocabulary-split table	tokens	1	1
dist_adamw_zero	new	one AdamW step with global gradient-norm clipping	parameters	1	2
dist_contrastive_loss	new	per-row contrastive (InfoNCE) loss against every other row	batch	2	2
dist_conv2d_halo	new	3×3 convolution with bias	height	1	2
dist_gemm_add_relu	KernelBench	$\mathrm{ReLU}(xW^\top + b_1 + b_2)$	input features	1	2
dist_gemm_gn_swish	KernelBench	GEMM, GroupNorm, Swish, multiply by a weight, Swish	output features	1	2
dist_matmul_gelu_softmax	KernelBench	row softmax of $\mathrm{GELU}(xW^\top + b)$	output features	1	2
dist_moe_dispatch	new	top-2 mixture of experts: softmax gate, $\mathrm{GELU}(xW_e^\top + b_e)$ per expert, gate-weighted sum	tokens, experts	1	2
dist_moe_router	new	top-1 mixture-of-experts routing with expert capacity: softmax, argmax, capacity count	tokens	1	2

Table 10: The nineteen bf16 models graded in MLScale, by level: eight distributed versions of KernelBench models (Ouyang et al., 2025) and eleven new models. *Split* is the dimension the manifest decomposes over GPUs, and k the degree of the work in it (Appendix E). Figure 6 labels dist_moe_router, dist_sdpa, dist_all_to_all_transpose and dist_softmax as MoE Router, SDPA, All-to-All and Vocab Softmax.

Intervention	Staged for the agent
Language skill packet	a skill packet: language, OpenMP and method pages for the setup's language; a hints page
Performance Toolkit	a skill packet: profiling tools; divide-and-conquer, profiling and compiler optimization-report pages (plus ROCm profiler on AMD)
CPF	the kernel's CPF (Budanaz et al., 2026) as starting source; a variant instead gives a CPF page and a tool returning the kernel's CPF (Qwen3.8-27B and OSS-120B), reported only in the setups table
Git reformulation	the kernel as a git repository with an issue
Blind	no score tool, one submission
Terse	a page that keeps every agent message brief (Harness20, and C and HIP on LLR-Focus40)
Harness	mini-SWE-agent or OpenHands in place of Claude Code, or AutoKernel (Jaber & Jaber, 2026), an add-on to Claude Code (Harness20)
RCCL hints	a skill packet: a page on RCCL collectives (MLScale)

Table 11: Interventions of §4.4.

with an x86-only construct (intrinsics headers, _mm*, __builtin_ia32_*, target attributes), builds only with AMD-only HIP features (rocPRIM or hipCUB, mask-less shuffles, AMD inline assembly, more than 48 KB static shared memory), or uses AMD-specific Triton options, cache hints or runtime calls. Table 3 gives the share skipped and the results per language.

D GRADING AND TIMING

D.1 SECRECY, TIMING AND EVASION

Secrecy. Two secret seeds fix the sampling: the secret score seed draws the inputs the score tool uses, and the secret submission seed draws the inputs of the final grade; and an image check asserts that no agent image holds either seed. Agents run unprivileged in private user, mount and PID namespaces and see only their workspace, the kernel and read-only prompt and skill files.

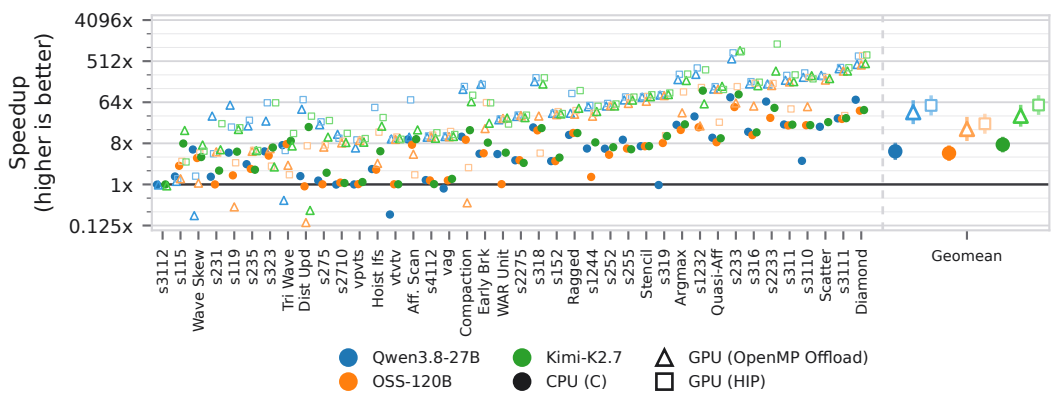

Figure 8: Per-kernel speedup over Numba on the LLR track, the answers of Figure 3. Geometric means over solved kernels for Qwen3.8-27B, Kimi-K2.7 and OSS-120B: CPU 5.3, 7.5, 4.9×; OpenMP target offloading 38.7, 32.3, 16.6×; HIP 54.5, 55.6, 21.6×.

Experiment	Dev.	Harness	Languages	Models	Control	Treatment	Budget
LLR-Focus40	CPU	CC	C, Fortran	Q, K, O	no skill packet	Language skill packet	12M, 4 h (K: 8 h)
LLR-Focus40	GPU	CC	HIP, Triton, OpenMP target offloading	Q, K, O	no skill packet	Language skill packet	12M, 4 h (K: 8 h; K HIP: 24M, 16 h)
LLR-Focus40	CPU	CC	C	Q, O	no skill packet	CPF page and tool (variant)	12M, 4 h
LLR-Focus40	CPU	CC	C	Q, K, O	no skill packet	CPF as starting source	12M, 4 h (K: 8 h)
LLR-Focus40 Blind	CPU	CC	C, Fortran	Q, K, O	open-submission	blind	12M, 4–5 h (K: 5–8 h)
LLR-Focus40 Blind	GPU	CC	HIP	Q, O	open-submission	blind	12M, 4–5 h
SciComp35	CPU	CC	C	Q, K, O	no skill packet	Performance Toolkit	60M, 20 h
Git-SciComp	CPU	CC	C	Q, K, O	bare kernel	git reformulation	60M, 20 h
Harness20	CPU	see text	C	Q, O	CC	mini-SWE-agent, OpenHands, AutoKernel	12M, 4 h (Q mini-SWE-agent, OpenHands: 24M, 8 h)
Terse	both	CC	C, HIP	Q, O	no instruction	brief-message instruction (Harness20: Q, C; LLR-Focus40: C, HIP)	12M, 4 h
MLScale	GPU	CC	HIP, RCCL	Q, K, O	no skill packet	RCCL hints (Q, O only)	24M, 6 h

Table 12: Setups. Both setups of a pair share model, language, device and kernels and differ only in *Treatment*. Q: Qwen3.8-27B, K: Kimi-K2.7, O: OSS-120B. CC: Claude Code. *Budget*: tokens and wall time per episode.

Timed inputs. Timed repetitions cycle through $D = 4$ seeded input draws, in one order for both sides; the warmup takes the first draw, so a result cached from the previous call fails the re-check.

Measurement. One thread per physical core and one GPU per grading process; the judge's clock stops after it synchronizes the timed code's device and OpenMP runtimes.

Crediting. A one-sided Mann–Whitney test (Hoefler & Belli, 2015) at $\alpha = 0.1$ in the direction of the medians credits confirmed speedups and confirmed slowdowns alike, and an input it does not confirm is credited $1\times$; it sets 215 of 8,764 correct timed inputs to $1\times$. The score tool runs the same code on one input and reports the fastest of five repetitions.

Plausibility. An input is suspect, and left out of S_i, if its median speedup exceeds $2000\times$ (host) or $16{,}000\times$ (device), its time is below declared bytes over twice the 5.3 TB/s HBM peak, or a device check fires; an answer whose inputs are all suspect is credited $1\times$. Section 4.2 describes the evasions these checks caught.

D.2 Correctness

Bands. NaN and $\pm\infty$ positions must agree, and each output's `atol` (Table 13) is raised to the reassociation floor

$$\texttt{atol}_{\text{eff}} = \max\big(\texttt{atol},\ \varepsilon_{\text{acc}}\sqrt{\ell}\,\|x_{\text{ref}}\|_\infty\big), \tag{5}$$

	fp64	fp32	fp16	bf16	fp8 (e4m3)	fp8 (e5m2)
`rtol`	10^{-9}	10^{-3}	10^{-2}	$3 \cdot 10^{-2}$	10^{-1}	$2 \cdot 10^{-1}$
`atol`	10^{-11}	10^{-5}	10^{-3}	10^{-2}	0.125	0.25

Table 13: Validation bands by precision. Other formats: $\texttt{rtol} = \sqrt{\varepsilon}$ clamped to $[10^{-11}, 0.25]$, $\texttt{atol} = \max(10^{-2}\,\texttt{rtol}, \varepsilon)$. Reduced formats accumulate in fp32 (Blanchard et al., 2020).

where ε_{acc} is the accumulation precision's machine epsilon and ℓ the largest product of an input's size symbols absent from the written output shape (K for $(M, K) \times (K, N)$). Kernels whose shapes do not fix ℓ use their largest input's element count; sequential scans declare their chain length. A configuration with $\varepsilon_{\text{acc}}\sqrt{\ell} \geq \texttt{rtol}$ is ungradeable, not incorrect.

Determinism. Two runs of one binary agree when their largest elementwise difference over $\varepsilon\sqrt{\ell}$ times the first run's max-norm is below 30, LAPACK's test-ratio threshold (Blackford & Dongarra, 1999).

D.3 Scoring rules

Efficacy statistics. The ratios of Equation equation 3 use per-kernel log changes, $d_i = \ln(S_i^{\text{after}}/S_i^{\text{before}})$ over $\mathcal{B}$ and $d_i = \ln(C_i^{\text{before}}/C_i^{\text{after}})$ over $\mathcal{K}$:

$$\rho = e^{\bar{d}}, \qquad 95\%\ \text{log-}t\ \text{interval}\ \exp\Big(\bar{d} \pm t_{0.975,\,N-1}\,\frac{\mathrm{sd}(d)}{\sqrt{N}}\Big), \tag{6}$$

with a two-sided paired t-test of $\bar{d} = 0$; no intervals below six pairs. The t interval assumes normal log-ratios across kernels; runtimes themselves are compared only by rank tests (§3). Solved counts are reported, not tested. Benjamini–Hochberg $q < 0.05$ over the speedup and cost tests of one panel is marked.

Example. Illustrative speedups, X unsolved:

$$S^{\text{before}} = (2,\ 4,\ \mathsf{X},\ 1,\ 8,\ 2,\ 4,\ 1), \qquad S^{\text{after}} = (4,\ 4,\ 2,\ 0.5,\ 16,\ 2,\ 8,\ 1). \tag{7}$$

Solved kernels rise from 7 to 8 ($\rho_R = 1.14$; $g = 1, l = 0$); over the seven both solve, the geometric-mean speedup rises from 2.44 to 2.97 ($\rho_S = 1.22$, interval $[0.75,\ 1.98]$, paired t-test $p = 0.36$).

Token counts. Engine cache counters are not used for Equation equation 4. Without a server count (killed process) the model's tokenizer counts the transcript, undercounting 2–4%. After compaction the rebuilt prompt counts as input.

E Distributed execution and scaling

Weak-scaling sizes. In Equation equation 2, P counts PEs, one MPI rank per PE. A manifest names the decomposed size symbols $N = (N_1, \ldots, N_d)$ and the degree k of the work in them, $W(sN) = s^k W(N)$ (a GEMM partitioning one row axis has $k = 1$). Fixed work per PE needs $s = P^{1/k}$, an integer m at $P = m^k$:

$$N_{P,j} \;=\; m\,N_{1,j}, \qquad j = 1, \ldots, d, \qquad P = m^k, \tag{8}$$

so $r_i(P) = P$, as in isoefficiency (Grama et al., 1993), HPCG and LULESH (Heroux et al., 2013; Karlin et al., 2013). At other P the symbols scale by $P^{1/k}$, rounded, and $r_i(P) = W(N_P)/W(N_1)$ is recorded. Table 14 gives the work and the exact weak-scaling PE counts per motif.

Ownership. A kernel declares each array's layout: a ScaLAPACK (Blackford et al., 1997) block, cyclic or block-cyclic distribution over a process grid, or replication. The agent writes all communication; the drivers only scatter inputs and gather outputs, so the correctness check and timed region keep their single-node definitions.

MLScale. The grade measures on 1, 2, 4, 8 and 16 GPUs across four nodes, four GPUs per node and one rank per GPU; a point is the median over runs of the slowest GPU's time, and B_i is the median of PyTorch (`torch.compile`) on one GPU at the base size. Input dimensions and per-GPU blocks are multiples of 64, the strong-scaling split dimension a multiple of 1024.

Motif	Work	Weak P	Strong decomposition
Dense LA / GEMM	$O(N^3)$	m^3	2D block-cyclic
2D stencil	$O(N^2T)$	m^2	1D / 2D block
3D stencil	$O(N^3T)$	m^3	3D block
SpMV / BFS / map-reduce	$O(\text{nnz})$ / $O(V+E)$ / $O(N)$	any (k=1)	row / graph partition
Direct N-body	$O(N^2)$	m^2	particle / domain
FFT	$O(N \log N)$	strong only†	slab / pencil transpose

Table 14: Work, exact weak-scaling PE counts and strong-scaling decomposition per motif. †No integer growth multiplies $N \log N$ exactly by P.

F TOKEN-COST CONVENTIONS OF PRIOR WORK

Where prior work states how it counts tokens, it sums every request's prompt, so in multi-turn agent sessions cached input dominates the count (Yu & Yang, 2026; Zhu et al., 2026) and varies with the serving engine's cache policy (Zheng et al., 2024; Lumer et al., 2026).

G REPRODUCING THE RESULTS

The only release is one anonymized Zenodo zip (`https://doi.org/10.5281/zenodo.22974128`, 31 MB) with a top-level `README.md` and two folders:

- `reproducibility/`, the artifact repository: the pipeline scripts (`download.sh`, `run_all.sh`, `stats.sh`, `fig*.sh`, `tab3.sh`, `lib/`, `tools/`), the measurements, extracted in `data/` and as `data.tar.zst` with one checksum per file in `DATA_SHA256SUMS`, the pooled answers in `work/`, every table and figure, the checksums of all outputs in `SHA256SUMS`, the submitted sources behind §4.2 and §4.6 (`case-studies/`) and every version of the skill packets (`skill_histories/`);
- `hpcagent-bench/`, the benchmark at the release tagged `paper-experiments`: benchmark code, experiment scripts and container definitions, with DaCe pinned through the `dace` dependency group of `pyproject.toml`.

User names, paths, host names and commit ids are replaced by placeholders, which changes no number.

G.1 REGENERATING EVERY FIGURE, TABLE AND NUMBER

This needs no GPU and no model access, only Linux with Python $\geq$ 3.12. From the unpacked zip:

```
cd reproducibility
python3 -m venv .venv && . .venv/bin/activate
pip install -e ../hpcagent-bench
./run_all.sh
```

The measurements ship extracted; `DATA_URL=file://$PWD/data.tar.zst ./download.sh` re-extracts them and fails unless every file matches `DATA_SHA256SUMS`. `run_all.sh` pools every answer at its final grade, runs the paired tests of §3, redraws every figure and table with the benchmark's plotting library, and checks all 54 outputs against `SHA256SUMS`. It ends with `OK: 54 files match SHA256SUMS`, or `sha256sum` names each file that differs. Table 15 lists the script behind each output.

Script	Output	In the paper
`stats.sh`	pooled answers, paired-test tables (`tables/*.csv`)	every figure
`fig2.sh`	`figures/efficacy_packets_and_scope.pdf`	Figure 3
`fig3.sh`	`figures/scope_row.pdf`	Figure 4
`fig4.sh`	`figures/cost_weighting.pdf`	Figure 5
`fig5.sh`	`figures/ml_scaling.pdf`	Figure 6
`fig8.sh`	`figures/cheating_per_kernel.pdf`	Figure 8
`tab3.sh`	`tables/gh200-transfer-table.tex`	Table 3

Table 15: Scripts of `reproducibility`; each runs after `stats.sh`.

G.2 RERUNNING AN EXPERIMENT

Platform. The experiments ran on a Slurm 25.05 cluster of MI300A nodes (four APUs per node, each 24 Zen 4 cores and 228 CDNA 3 compute units sharing 128 GB HBM3) with the Enroot container runtime; the GH200 regrade ran on GH200 nodes (four superchips per node, each 72 Neoverse V2 cores and one H100). The CPU experiments need CPU nodes only; the GPU experiments need AMD GPUs. All toolchains live in the images: ROCm 7.2.0 in the judge, agent and model-server images (SGLang for Qwen3.8-27B and Kimi-K2.7, vLLM for OSS-120B).

One-time setup. Two jobs of `hpcagent-bench/containers/cluster/ce-images` prepare the cluster (commands in its `README.md`): `build_and_verify.sbatch` builds the judge, agent and model-server images and checks each against the device, and `inference/fetch_weights.sbatch` downloads the model weights. The CPF setups also read their pre-rendered forms from `HPCAGENT_BENCH_CPF_PRERENDER_DIR`.

One command per experiment. Set `SCRATCH` to the directory that holds `hpcagent-bench/` and its virtual environment. From `hpcagent-bench/experiments/`, the command of Table 16 reruns one experiment end to end, with the budgets of Table 2; `SUBMIT=0` prints the plan without submitting. It submits one Slurm job per setup, and each job starts the model server, the judge and the agents in the fixed images. Every job ends within its wall limit of at most 23 h, so graded results are back within a day when the cluster starts an experiment's jobs together.

Experiment	Command (from `experiments/`)
LLR-Focus40, CPU	`SUBMIT=1 ./submit-cpf-llr40.sh`
LLR-Focus40, GPU	`SUBMIT=1 ./submit-gpu-llr40.sh`
LLR-Focus40 Blind	`SUBMIT=1 ./submit-llrblind.sh`
Git-SciComp	`SUBMIT=1 ./submit-git-scicomp.sh`
SciComp35	`SUBMIT=1 ./submit-scicomp-perf-playbook.sh`
Harness20	`SUBMIT=1 ./submit-harness-focus20.sh`
Terse (Harness20)	`SUBMIT=1 ./submit-harness20-caveman.sh`
MLScale	`SUBMIT=1 PACKET= ./submit-mlscale.sh`[†]

Table 16: One command per experiment. [†]Run again with `PACKET=dist-rccl-amd` for the RCCL-hint setups.

Grading. Every submit script chains a grading job on each agent job (`finalize_grade.sbatch`), which applies the current grading rules, the ones behind every number in this paper. The final grade re-times each answer on inputs drawn from the secret submission seed (Appendix D); items a grading job does not reach within its wall time are planned again by the next one (`regrade_rest.py`). An answer whose final re-timing yields no valid measurement keeps its last valid grade, and an answer without a final grade keeps the grade the judge gave during the episode. Kernels a setup never ran are excluded from that setup; a kernel that ran and failed counts as unsolved, so it enters the success rate and the token cost but not ρ_S. The multi-node scaling grade of the ML track is scheduled as its answers arrive.

From finished runs to figures. Each job writes its run directory to `$SCRATCH/hpcagent-bench-runs/<experiment>-<date>/<job>`. On the cluster, from `reproducibility/`, `PULL=0 MIRROR=$SCRATCH ./download.sh --from-cluster` rebuilds `data/` from those directories; elsewhere, `./download.sh --from-cluster` first mirrors them over ssh from the host set in `tools/cluster.env` (copied from `tools/cluster.env.example`). `./run_all.sh` then redraws every figure, table and number from the new data, and its checksum check names the outputs that changed. Sampling makes single episodes differ between reruns; the paired intervals are the quantity we expect to reproduce.