SWE-Marathon: testing Fable 5

An independent local run of an ultra-long-horizon SWE benchmark, asking three questions: how does the model compare to released Opus 4.8, does multi-agent orchestration help, and does aggressive prompting help?

2026-06-07 · n=1 per cell · not an official submission · Harbor harness on one x86 box

The benchmark

SWE-Marathon (Abundant AI, 2026-06-05) measures whether an agent can **autonomously complete project-scale software work** — build a C compiler, reimplement a Java language server, hand-optimize a VLIW kernel — tasks the authors estimate at **40–400 human-expert hours**, with a 2–10 h agent budget. Scoring is **binary**: 1.0 only if *every* verifier check passes; a continuous *partial score* is recorded as a diagnostic. The authors' 13-config sweep tops out at **26% pass@1** (vs an 84% human-oracle ceiling); many flagship tasks sit at **0% across every config**.

Method & honest framing

We ran `claude-fable-5[1m]` in **stock Claude Code**. The authors' sweep includes a **Claude Code + Opus 4.8** config with the identical scaffold and *no* custom system prompt, so swapping only the model is a legitimate controlled A/B. Read everything as a **replication probe, not a leaderboard claim**: n=1 here vs n=5 for the authors; partial score is a diagnostic, not a pass; and one variable is uncontrolled — Claude Code `v2.1.123` (authors) vs `v2.1.167–168` (us), so the default scaffold isn't byte-identical.

Result 1 — Default config vs. Opus 4.8 (same scaffold)

TASK	OPUS 4.8 (N=5) BINARY / PARTIAL	FABLE 5 (N=1) BINARY / PARTIAL	READ
wasm-simd	5/5 · 1.00	PASS · 1.00	Replicates the pass.
rust-c-compiler	0/5 · .981/.989	0 · .9888 (884/894)	Par — same near-miss band.
rust-java-lsp	0/5 · .799/1.0	0 · 1.0 vis / .993 hold	Par — perfect visible, 2 holdout misses.
ruby-rust-port	0/5 · .589/.985	0 · ~0 (quit early)	Trails.
vliw-kernel-opt	4/5 · 1.00	crash	Trails — Opus passes; Fable 5 hit an output-cap crash.
find-network-align	1/5 · .914	refused	Trails — API policy refusal (bio task).
biofabric-rewrite	0/5 · .771/.985	refused	Trails — API policy refusal (bio task).

"Opus 4.8" = authors' Claude Code + Opus 4.8, n=5: trials-passed / (mean-partial / best-partial).

> **Read:** on tasks it completes, Fable 5 roughly **matches** Opus 4.8's correctness (and at fewer tokens / less wall-clock — e.g. wasm-simd passed on 4.4M tokens vs Opus's 7.6M+). But it **loses three tasks to operational failures** Opus 4.8 did not hit: one output-token-cap crash and two reproducible policy refusals on benign computational-biology tasks. No binary pass except wasm-simd.

REPRO. Harbor 0.8.0 (RishiDesai fork) · abundant-ai/swe-marathon @61a8babb · Claude Code v2.1.167–168, stock, bypassPermissions, OAuth · CLAUDE_CODE_MAX_OUTPUT_TOKENS=64000 · Hetzner cpx51 16vCPU/32GB ubuntu-24.04 x86 · 7 of 20 tasks (CPU-only, open-internet) · nop=0.0 / oracle=1.0 validated. Cost: model pricing omitted; comparisons use correctness, tokens, wall-clock. Contamination: training cutoff unknown, benchmark released 2026-06-05; no decontamination claim.

Result 2 — Does more orchestration help? (ultracode)

We re-ran two tasks with Claude Code's multi-agent mode (the `ultracode` keyword), which fans the work out to subagents — confirmed real: up to **25 subagents** across parallel workflows, in both headless and a custom interactive harness.

TASK	DEFAULT (1 AGENT)	ULTRACODE HEADLESS	ULTRACODE INTERACTIVE
rust-c-compiler	.9888 (884/894)	.9911 (886/894) · 9 ag	.9888 (884/894) · 7 ag
rust-java-lsp	1.0 vis / .993 hold	1.0 vis / .993 hold · 25 ag (after integrity-scan fix)	

Verdict: no gain. On rust-c-compiler every config lands in an 883–886/894 band — pure run-to-run noise; 7–25 subagents and 600+ turns reproduce what one agent does in 207. On rust-java-lsp, ultracode first scored **0** — its subagents mined the fixture into multi-MB scratch files that tripped the "large fixture-derived data" integrity scanner (a *false positive*: it built a real LSP). Told to delete scratch before submitting, it scored **identically to one default agent. Fan-out doesn't improve correctness here** — and a thorough multi-agent style can actively trip reward-hacking defenses.

Result 3 — Does aggressive prompting help? (`/goal`)

Since this isn't an official submission, we can prompt however we like. We used Claude Code's `/goal` command — *"a goal Claude checks before stopping"* — as a stop-gate: *not done until the verifier shows a perfect score; re-run before stopping; never settle for a near-miss.*

TASK	NEUTRAL RUN	WITH /GOAL	GAP TYPE
vliw-kernel-opt	crash / DNF	PASS 1.0 (1187 cyc)	**Effort** — fixed
rust-c-compiler	.9888 (884/894)	.9888 (884/894)	**Capability** — not fixed

The key finding — a clean dissociation. `/goal` turned vliw from a crash into an outright **binary pass** (kernel optimized to 1187 cycles, under the 1250 gate — a task Opus 4.8 only gets 4/5). But on rust-c-compiler it changed nothing: 533k tokens and two hours of relentless pushing returned the *same 884/894 with the same 10 failing tests* (C23 compound literals, unicode identifiers, signed-overflow `fwrapv`). **Pressure fixes effort gaps — the agent stopping short — but not capability gaps, where the model genuinely can't implement the remaining cases.** The identical failing-test names confirm a deterministic knowledge wall, not variance.

Bottom line

- **Capability:** Fable 5, stock, roughly matches released Opus 4.8 on tasks it finishes (cheaper in tokens/time), but regresses via an output-cap crash and reproducible benign-bio refusals that Opus 4.8 handled.
- **More agents ≠ better.** Ultracode fan-out (up to 25 agents) gives no correctness gain over one agent, and its thoroughness can trip integrity scanners.
- **Smarter pressure helps selectively.** A stop-gated goal converts "stopped short" failures into passes (vliw), but can't move a true capability wall (rust-c-compiler). This tells you *which* near-misses are worth re-prompting and which need a better model.

Independent run, single x86 box, Fable 5 via Max-subscription OAuth. Not an official SWE-Marathon submission (the benchmark has no external submission process). Author figures from swe-marathon.org site data, 2026-06-06. Binary rewards are 0 except the two passes noted (wasm-simd default; vliw under /goal).