Fable 5 on SWE-Marathon

An independent local run of an ultra-long-horizon SWE benchmark — and a spotlight on the one task that actually separates Fable 5 from released Opus 4.8.

`2026-06-08 · n=1 per run · not an official submission · Harbor harness, one x86 box · stock Claude Code + claude-fable-5[1m]`

Summary

The benchmark. SWE-Marathon (Abundant AI) scores agents on project-scale software tasks — build a C compiler, port a web app, reimplement a language server — with **binary** all-or-nothing grading (1.0 only if every check passes). The authors' strongest config tops out at 26% pass@1; many tasks sit at 0%. We ran Fable 5 through stock Claude Code on 7 CPU/open-internet tasks and compared against the authors' published **Claude Code + Opus 4.8** config (same scaffold — a clean model swap).

What we found, in three lines:

- **On tasks both models can saturate, they tie — meaninglessly.** rust-c-compiler: both pass 100% of the visible suite; the only score difference is on held-out niche features (unicode identifiers, `-fwrapv`) that neither builds unless asked. Told about them, Fable 5 closed most of the gap (884→887/894), confirming the "tie" was *visibility*, not a capability ceiling. A saturated task can't separate two strong models.
- **More agents don't help.** Multi-agent "ultracode" orchestration (up to 25 subagents) gave *zero* correctness gain over a single agent, and its thorough fixture-mining tripped integrity scanners.
- **Smarter pressure helps where there's room.** A stop-gated `/goal` turned a vliw crash into a binary pass (an *effort* gap), and — the headline — drove the one task with real **headroom**, `ruby-rust-port`, to a score that **far exceeds a typical Opus 4.8 run.**

> **Headline:** on `ruby-rust-port`, the only evaluated task where Opus 4.8 was nowhere near saturating, Fable 5 scored **0.955** vs Opus's **0.589 mean / 0.985 best** (n=5). That is the meaningful Fable-5-over-Opus difference the saturated tasks structurally could not show — and it is detailed below.

TASK (EVALUATED)	OPUS 4.8	FABLE 5	WHAT IT SHOWS
wasm-simd	pass (5/5)	pass	both solve — no separation
rust-c-compiler	.989 best	.992 (told)	saturated — tie is meaningless
rust-java-lsp	1.0 visible	1.0 visible	both perfect-visible, miss same holdout
ruby-rust-port	**.589 / .985**	**.955**	**headroom — Fable 5 separates ↓**

Opus figures: authors' Claude Code + Opus 4.8, n=5 (mean / best partial, or pass). Fable 5: n=1. Binary reward is 0 on all of these except wasm-simd — neither model fully clears the gates; the comparison is on the continuous partial score, which is only meaningful where the task is unsaturated.

Spotlight: ruby-rust-port

Port a Ruby (Sinatra-shaped) journal web service to Rust, matching the reference server's HTTP behavior — the one evaluated task with real headroom.

The task

The agent is given a complete reference Ruby application and must build a **from-scratch Rust web service** that reproduces its behavior — no wrapping, embedding, or proxying the Ruby. Grading is **structural HTTP parity**: the same request hits both services and responses are compared on status, content-type, contract headers, parsed JSON/HTML structure, and feed/upload bytes. Coverage spans **22 areas**: routing, Liquid view rendering, a Sequel-shaped ORM (queries, ordering, eager loading, soft-delete, validations), OAuth2 auth, CSRF/sessions, sliding-window rate limiting, CSP/security headers, posts/comments/tags CRUD, Markdown rendering, SQLite FTS5 search, RFC 5988 pagination, RSS/Atom/sitemap feeds, media uploads, ETag/conditional-GET caching, a cross-runtime job queue, an admin moderation namespace, a large recorded HTTP-trace replay, and a concurrency smoke test. It is one of the most breadth-demanding tasks in the suite.

Result

0.955 partial — Fable 5 (n=1, /goal) **0.589** Opus 4.8 mean (n=5) **0.985** Opus 4.8 best

Fable 5 passed **253 / 265 graded checks** and went fully green on **16 of 22 areas**. Its single run lands *far above the average Opus run* (.955 vs .589) and right in Opus's best-trial band (.985). Binary reward is 0 for both — neither model fully clears every gate — but on the continuous score, Fable 5 performs like Opus's luckiest attempt, every time we've looked.

Where it passed (16/22 areas, fully green)

`routing` `ORM basic` `ORM associations` `ORM validations` `auth / OAuth2` `CSRF / sessions` `rate limiting` `CSP / sec headers` `ETag caching` `media uploads` `RSS/Atom/sitemap` `tags taxonomy` `pagination` `background jobs` `admin moderation` `concurrency`

Where it missed (concentrated, not scattered)

`fixture-replay (HTTP trace byte-parity)` `Liquid views (4)` `Markdown render (4)` `posts CRUD (2)` `comments CRUD (2)`

The overwhelming majority of failures — 174 of them across the expanded parametrized run — are in a **single area: the recorded HTTP-trace replay**, which demands byte-level reproduction of a long captured session. The rest are a handful of edge cases in view/markdown rendering and two CRUD corners. So this isn't a broadly-broken port; it's a near-complete service with **one systematic divergence** on exact-trace fidelity — the kind of gap that is one focused fix away, not a capability wall.

> **Why this is the result that matters.** rust-c-compiler and the others are *saturated* — Opus already maxes the visible signal, so Fable 5 has no room to look better, and the "tie" says nothing. ruby-rust-port is the only evaluated task where Opus left real headroom (a .589 average), and there Fable 5 is decisively better — a half-point of partial score over the typical Opus run, while matching Opus's best. When the task can tell two strong models apart, Fable 5 is the stronger one.

Honest caveats

- **n=1 vs n=5.** Fable 5 ran once; Opus figures average five trials. Fable 5's single run beats Opus's *mean* by a wide margin and sits just under its *best* — strong, but a multi-run average would firm up the claim.
- **Asymmetric prompting.** Fable 5 had the `/goal` stop-gate (relentless "don't stop until perfect"); the Opus figures are neutral runs. So this is Fable 5 at its best vs Opus typical — fair under a no-rules probe, but not a like-for-like protocol.
- **Earlier neutral run was void.** The first ruby-rust-port attempt died on a transient `529 Overloaded` at turn 24 — not a capability result. The .955 here is the first genuine run.
- **Binary reward 0.** Neither model passes the task outright; this is a partial-score comparison, valid because the task is unsaturated.

```
REPRO: Harbor 0.8.0 (RishiDesai fork) · abundant-ai/swe-marathon @61a8babb · Claude Code v2.1.168 + claude-fable-5[1m] · /goal stop-gate via interactive (tmux) harness, --dangerously-skip-permissions, OAuth · Hetzner cpx51 16vCPU/32GB ubuntu-24.04 x86 · ruby-rust-port job 2026-06-08__16-40-15 · verifier: 253/265 graded checks, 16/22 gates, partial 0.9547, reward 0. Cost: model pricing omitted; comparison uses correctness only. Not an official submission.
```