Route Coding Agents by Task, Not Brand

Coding Agents Alpha Tracker

2026-07-19

Route Coding Agents by Task, Not Brand

By Coding Agents Alpha Tracker • July 19, 2026

Today's practical edge is task-aware model routing: use inexpensive persistent workers for execution, and reserve higher-judgment models for planning, review, design, and merge-quality diffs. Also covered: a Kimi K3 Claude Code configuration and Claude Code's move to Rust Bun.

TOP SIGNAL

Route models by the job, then split planning from execution. In first-hand daily use, Theo starts with Soul for most work but moves to Fable when a change must merge cleanly, the design matters, or the problem is unusually complex; for multi-agent work, he has Fable plan and judge while Soul completes assigned pieces. [1] Codex's separate, name-addressable threads offer a lightweight version of that orchestration pattern. [2]

TRY THIS

- **Adopt a two-tier model router.** Send quick local changes, debugging, terminal work, and long-running goals to Soul; escalate to Fable for UI exploration, simplifying a diff, verifying another agent's work, or producing a PR you expect to merge. [1]
- **Make the planner's output the worker's contract.** Ask Fable to break work into small, explicit pieces, then give Soul a specific implementation assignment for each piece. In Codex, tell it to fan out into separate threads for the task; reference another thread by name when handing work back for coordination. [1, 2]
- **Run Kimi K3 from Claude Code on PowerShell.** Jason Zhou's setup points Claude Code at Moonshot's Anthropic endpoint, assigns `kimi-k3[1m]` to the main and subagent model slots, sets a 1,048,576-token auto-compact window, then starts `claude`:

```
$env:ANTHROPIC_BASE_URL="https://api.moonshot.ai/anthropic"
$env:ANTHROPIC_AUTH_TOKEN="YOUR_MOONSHOT_API_KEY"
$env:ANTHROPIC_MODEL="kimi-k3[1m]"
$env:ANTHROPIC_DEFAULT_OPUS_MODEL="kimi-k3[1m]"
$env:ANTHROPIC_DEFAULT_SONNET_MODEL="kimi-k3[1m]"
$env:ANTHROPIC_DEFAULT_HAIKU_MODEL="kimi-k3[1m]"
$env:ANTHROPIC_DEFAULT_FABLE_MODEL="kimi-k3[1m]"
$env:CLAUDE_CODE_SUBAGENT_MODEL="kimi-k3[1m]"
$env:ENABLE_TOOL_SEARCH="false"
$env:CLAUDE_CODE_AUTO_COMPACT_WINDOW="1048576"
$env:CLAUDE_CODE_EFFORT_LEVEL="max"
claude
```

[3]

- **Treat high-autonomy goals as destructive-capable.** Theo cautions that Soul on Ultra can pursue a goal aggressively enough to delete files or databases; he cites a case where `rm -rf` wiped a developer's user directory and in-progress code. Do not give that mode unchecked access to work you cannot afford to lose. [1]

WHAT SHIPPED

- **Claude Code v2.1.181+: Rust Bun underneath.** Claude Code now uses Bun's Rust port; Jarred Sumner reported a **10% Linux startup improvement**. Simon Willison found `Bun v1.4.0 (macOS arm64)` embedded in the binary—newer than Bun's then-public v1.3.14—and extracted 563 Rust source filenames. [4]
- **Fable vs. Soul: modest benchmark gap, large cost delta.** Theo reports Cursor Bench results of **70% for Fable** versus **67% for Soul**, while citing roughly **$17/task** for Fable Max and **$5/task** for Soul. His practical distinction is code shape: Soul may write far more than needed, while Fable more often produces the smaller change he is willing to merge. [1]

GO DEEPER

- **0:08–2:15 — Why experienced users split between Fable and Soul.** Theo opens with competing practitioner preferences, then frames why task fit matters more than declaring a universal winner.

Fable 5 vs GPT-5.6 (0:08)

- **25:05–26:37 — "Passes tests" is not the same as "ready to merge."** Theo's strongest argument for Fable is its tendency toward minimal diffs rather than excessive code—a useful lens for evaluating your own agent harness.

Fable 5 vs GPT-5.6 (25:05)

Editorial take: the durable workflow is not loyalty to one model—it is explicit routing, planner/worker separation, and tight control over what autonomous agents can touch. [1]

Sources

1. Fable 5 vs GPT-5.6
2. X post by @HamelHusain
3. X post by @jasonzhou1993
4. Claude Code uses Bun written in Rust now