WHITE PAPER · MODEL ROUTING IN AGENTIC CODING

Cheapest per token is not cheapest per task

A pre-registered test of model routing for agent handoffs on Claude models, and what the wider research says about why routing pays off less than its price list suggests.

76%
lower cost per completed task when the parent picks the worker model, versus defaulting to Opus

23%
cheaper again: a fixed “always Sonnet” default beat the router, at the same 100% completion

2.4×
Haiku’s cost per completed task versus Sonnet, despite half the per-token price

1.3%
the most a perfect hindsight router could have saved over always-Sonnet on this task set

Andrew Kaiserauer

September 2026 · Version 1.0 (confirmatory stage 1 results)
Data: 780 graded agent sessions, 65 tasks × 3 trials × 4 policies, $202 list-price spend
Harness and raw data: `github.com/kornsour/model-routing`

Executive summary

Model routing promises to cut AI spend by sending easy work to cheap models and hard work to expensive ones. The promise is usually argued from per-token price lists. What an organization actually pays for is completed work, and in agentic coding a task is dozens of model calls whose number depends on which model runs them.

This paper reports a pre-registered experiment that measured **cost per completed task** for 65 coding tasks handed off from a Claude Opus session to a fresh worker agent, under four dispatch policies, three trials each, graded by hidden tests with no LLM judge. It then places the result against the published research on routers, including the commercial tools (OpenRouter, GitHub Copilot Auto, Cursor, AWS Bedrock, Azure) that have made routing mainstream.

FINDINGS

1. **The pre-registered hypothesis held.** Letting the parent session name the worker model while writing the handoff brief cut cost per completed task by 76% (95% CI 70% to 80%) versus spawning every worker on Opus, with no loss in completion (100% vs 99%). Billing the router's entire turn still leaves a 50% saving.
2. **But a fixed default did better.** Always spawning on Sonnet completed 100% of tasks at $0.099 each, 23% below the router. The router chose Sonnet 87% of the time, and every departure from Sonnet added cost. The saving came from not defaulting to the most expensive model, not from choosing per task.
3. **The cheapest model was not the cheapest per task.** Haiku completed 84% of tasks at $0.239 per completed task, 2.4× Sonnet. It took 3.5× the turns and re-read 7× as much cached context, so it cost more than Sonnet even on the easy tasks it passed.
4. **There was almost nothing left to route.** A perfect hindsight oracle would have saved only 1.3% over always-Sonnet. No task in the set required Opus. Routing can only earn money when some tasks need the expensive model and others don't; this workload had one clear winner.
5. **The literature agrees once you separate settings.** Headline router savings of 45% to 98% come from single-turn chat and QA benchmarks with a 20× or wider price gap. Independent re-evaluations find routers often fail to beat simple baselines, and the newer agentic studies find routing harder: prompt caches are per model, weaker models take more turns, and task difficulty is often invisible until the agent has explored the code.

Is it the models or the harness?

Both, but neither is a defect. Three things compress routing's payoff here. **The price ladder is narrow**: Sonnet 5 costs 2× Haiku 4.5 per token and Opus 5 costs 2.5× Sonnet, far from the GPT-4 versus Mixtral gap behind the famous RouteLLM numbers. **Tokens per task is a trained property**, and on this workload the mid-tier model was the most efficient, finishing in about half of Opus's turns and a quarter of Haiku's. **The agent loop multiplies turn count**: each turn re-reads the growing context, so extra turns cost more than linearly, and prompt caches cannot be shared across models. Different training (cheap models that stop when done or ask for help, in-model effort control) and a different harness (routing after a short exploration, a real verifier for cascades, shorter cache lifetimes for short workers) could each open more headroom. Section 6 covers what the evidence says about each.

What to do with this

- **Set the handoff default to the mid-tier model**, not the parent's model. This alone captured almost all of the available saving.
- **Judge any router, commercial or custom, on cost per completed task against the best fixed default**, not against the most expensive model. Beating "always Opus" is easy; beating "always Sonnet" is the real test.
- **Route at clean context boundaries** (new subagents, new tasks), never mid-conversation, and prefer effort control within one model over switching models when the context is warm.

1 Why routing looks like free money

The routing pitch is arithmetic on a price list. If a frontier model costs 5× a small one and 70% of requests are easy, sending those to the small model should cut the bill by more than half. Early academic results supported this: RouteLLM reported cost reductions of over 85% on MT-Bench while keeping 95% of GPT-4's quality[1], FrugalGPT reported up to 98%[2], and AutoMix over 50%[4]. Commercial products followed. AWS markets Bedrock Intelligent Prompt Routing as cutting costs "by up to 30% without compromising on accuracy"[25], GitHub gives a 10% discount for using Copilot's Auto model selection[27], and OpenRouter, Cursor, Windsurf and Azure all ship automatic model selection.

Agentic work breaks three assumptions behind that arithmetic.

1. **A task is many calls, and the count depends on the model.** A coding agent reads files, runs tests, edits and re-runs. A weaker model that takes three times the turns can cost more in total even at half the per-token price.
2. **Each call re-reads the conversation so far.** Prompt caching makes this cheaper, but caches are per model. Anthropic's documentation is explicit that switching models recomputes the entire request[20], and the Claude Code team has written that switching a long Opus conversation to Haiku "would actually be more expensive" than letting Opus answer, because the Haiku cache must be rebuilt[19].
3. **Difficulty is often invisible up front.** A one-line bug report can hide a one-line fix or a multi-module refactor. A router reading only the prompt cannot tell which[13].

The natural place to route in an agent system is therefore the **handoff**: the moment a parent session spawns a fresh worker (a subagent, a task chip, a background job) with a written brief. The worker starts with an empty cache whatever model it runs on, the parent has just read the relevant context, and the brief is the worker's whole input. Asking the parent to also name the model costs almost nothing. This study tests whether that cheap decision pays.

2 Study design

2.1 Question and pre-registration

The primary hypothesis (H-D1) was fixed before the confirmatory run, along with the metric, the non-inferiority margin, the policies, the trial count, the randomization seed and a hash of the task set:

Spawning with the model the parent recommends while it writes the brief (policy C1_inline) has a lower cost per completed task than spawning with the parent's own model (policy B), with a pass rate no more than 10 percentage points worse.

The report tool marks a run confirmatory only when every registered field matches. Everything else in this paper, including the comparison against always-Sonnet, is labeled exploratory.

2.2 Policies

Policy	What happens at the handoff	Cells
B (control)	Fresh worker session on the parent's model, Claude Opus 5. This is the status quo for most agent harnesses.	195
C1_inline (treatment)	The parent writes the brief and ends with a one-line pick from a menu (Haiku, Sonnet, Opus at low effort, Opus). The worker runs on that pick with the same canned brief as B, so the two arms differ only in model.	195
static_sonnet	Fresh worker on Claude Sonnet 5 for every task.	195
static_haiku	Fresh worker on Claude Haiku 4.5 for every task.	195
oracle	Computed afterwards: for each task and trial, the cheapest static model that passed. Not deployable; it bounds what any router could save.	(195)

Four policies × 65 tasks × 3 trials = 780 graded cells. Sessions ran strictly sequentially in seeded randomized blocks, with a 40-turn cap per session for every model.

2.3 Tasks and grading

The 65 tasks live in three purpose-built Python repositories (invoicing, log processing, a notes CLI): 29 bug fixes, 13 features, 5 config, 5 performance, 4 migrations and a handful each of docs, refactor and tests. A calibration run beforehand (66 tasks on all three models, two trials, $110) measured difficulty instead of guessing it: 50 tasks were **easy** (Haiku passed every trial) and 15 **medium** (Haiku passed some). One task that no model could pass was excluded before registration as a specification gotcha. Notably, **no task was passed only by Opus**.

A cell passes only when the hidden tests pass, the visible tests still pass, and no file outside the task's allowed paths changed. Grading is deterministic; there is no LLM judge.

To check realism, the brief format was compared with 562 real handoffs harvested from the author's own Claude Code transcripts. The synthetic briefs match on structure (they name files and span several) but are shorter (median about 270 tokens versus about 500 for real task chips) and more uniformly specified. In real use, parents already routed subagents between Sonnet and Opus roughly 9 to 1 and never picked Haiku.

2.4 Models and pricing

Model (as resolved)	Input	Cache read	Cache write, 1 h	Output	Relative to Sonnet
claude-haiku-4-5	$1.00	$0.10	$2.00	$5.00	0.5×
claude-sonnet-5	$2.00	$0.20	$4.00	$10.00	1×
claude-opus-5	$5.00	$0.50	$10.00	$25.00	2.5×

List prices per million tokens as used by the harness. Claude Code writes its prompt cache with the 1-hour lifetime, billed at 2× input. All 1,171 sessions reported a resolved model id and a CLI cost that matched the list-price calculation.

All figures are list API prices, which is what an API-key user pays. Sessions ran through the Claude Code CLI (version 2.1.278) on a subscription login; a subscriber pays a flat fee and consumes a usage allowance instead, and relative costs should carry over to how fast that allowance runs down.

3 Results

3.1 The primary test passed by a wide margin

Policy	Pass rate (95% CI)	Cost per completed task (95% CI)	Mean turns	Errored cells
B: always Opus	99% (98 to 100)	$0.528 ($0.454 to $0.599)	14.6	0
C1_inline: parent picks	100% (100 to 100)	$0.129 ($0.104 to $0.167)	9.3	0
static_haiku	84% (76 to 91)	$0.239 ($0.200 to $0.293)	26.3	36
static_sonnet	**100% (100 to 100)**	**$0.099 ($0.090 to $0.109)**	**7.4**	**0**
oracle (hindsight)	100%	$0.098	n/a	n/a

Intervals are 95% task-clustered bootstrap (all trials of a task resampled together). Every errored cell was a Haiku session that hit the 40-turn cap; it was graded on whatever state it left.

Against the Opus default, parent-picked routing saved **76% per completed task (95% CI 70% to 80%, p < 0.001)** and changed the pass rate by +0.5 points (CI 0 to +1.5), comfortably inside the 10-point margin. The pre-specified subgroup of 15 medium tasks, where Haiku sometimes fails, showed an 80% saving at a 100% pass rate; the parent never picked Haiku there. The router recovered 93% of the oracle's saving over Opus.

Two sensitivity checks leave the verdict unchanged. Billing the parent's entire brief-writing turn to routing, which overstates the cost because B also needs a brief, cuts the saving to 50% (CI 43% to 55%). Excluding errored cells changes nothing, because neither H-D1 arm had any.

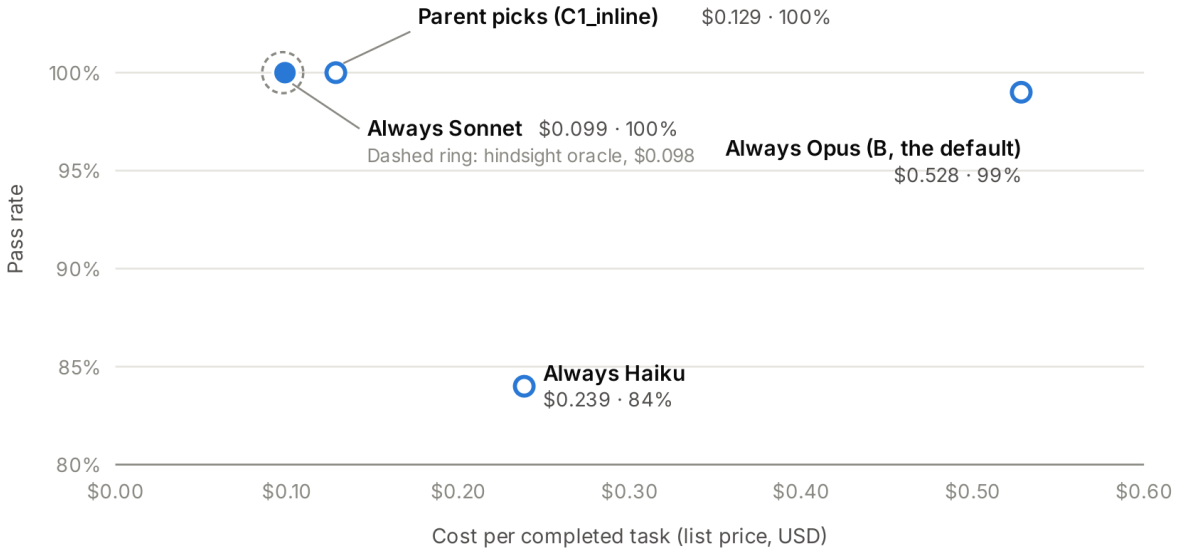

Figure 1. Cost per completed task against pass rate. Always-Sonnet sits almost on top of the hindsight oracle and dominates every other policy, including the router. Always-Haiku is both cheaper-per-token and worse on both axes than Sonnet.

3.2 But the router lost to a fixed default

The comparison readers will ask about was not pre-registered, so it is exploratory: **against always-Sonnet, the router cost 30% more per completed task** (task-clustered 95% CI +9% to +59%) at the same 100% completion. Put the other way, always-Sonnet was 23% cheaper than the router.

Router's pick	Share of cells	Passed	Mean cost per task	Compared with Sonnet on the same cells
Sonnet	87%	169 / 169	$0.106	Same model; small overhead from the pick
Haiku	10%	19 / 19	$0.062	Slightly dearer than Sonnet's $0.055 on those cells
Opus	4%	7 / 7	$0.848	Sonnet passed these too, at a fraction of the cost

Router picks from the C1_inline arm. Costs include the marginal cost of the pick.

The parent made sensible-looking choices: it sent the simplest tasks to Haiku and a few it judged risky to Opus. Both kinds of departure raised cost. Haiku's "easy" wins were no cheaper than Sonnet's, and the Opus picks bought insurance the tasks did not need.

3.3 Where Haiku failed

Measured difficulty	B (Opus)	C1_inline	Haiku	Sonnet	Haiku cost / pass	Sonnet cost / pass
Easy (50 tasks)	100%	100%	95%	100%	$0.177	$0.089
Medium (15 tasks)	98%	100%	47%	100%	$0.657	$0.132

By task category, Haiku's losses concentrated in bug fixes (76% pass), migrations (67%) and performance work (67%); it passed every docs, feature, refactor and tests task.

Even on easy tasks, where Haiku passed 95% of the time, it cost twice as much per completed task as Sonnet. On medium tasks it cost five times as much, because it failed half the time and burned its full turn budget doing so.

4 Why the cheapest model cost the most per task

The per-session token records explain the inversion. Almost none of the cost is fresh input. It is cache reads (the conversation re-read on every turn), cache writes (new context stored at the 1-hour rate), and output.

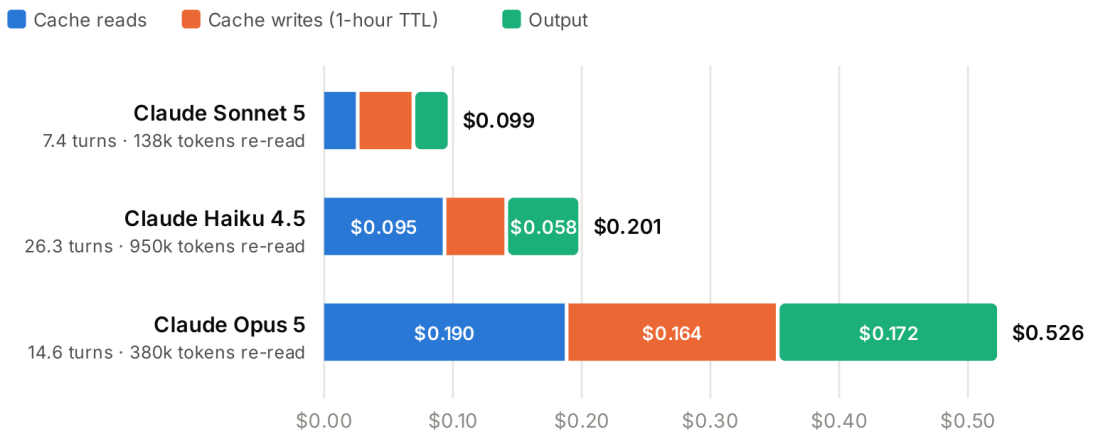

Figure 2. Mean worker session cost by token type (static arms, 195 sessions per model). Haiku takes 3.5× Sonnet's turns but re-reads 6.9× the tokens, because every extra turn re-reads a longer context. Opus writes about 1.5× and outputs about 2.4× what Sonnet does, on top of a 2.5× per-token price.

Per worker session	Turns	Tokens re-read from cache	Tokens written to cache	Output tokens	Cache hit rate	Cost
Sonnet 5	7.4	137,669	10,820	2,811	93%	$0.099
Haiku 4.5	26.3	950,263	23,933	11,500	98%	$0.201
Opus 5	14.6	379,632	16,398	6,871	96%	$0.526

Means over all 195 worker sessions per model in the static and B arms, including failed sessions.

Two effects compound. First, **turn count is superlinear in cost**. If each turn adds roughly the same amount of context, total re-read tokens grow with the square of the turn count, so 3.5× the turns gives about 7× the reads. A high cache hit rate does not rescue this; Haiku's was the highest of the three at 98%. Caching makes each re-read cheap, not free. Second, **the per-token price gap is small next to the per-task behavior gap**. Figure 3 puts the two side by side.

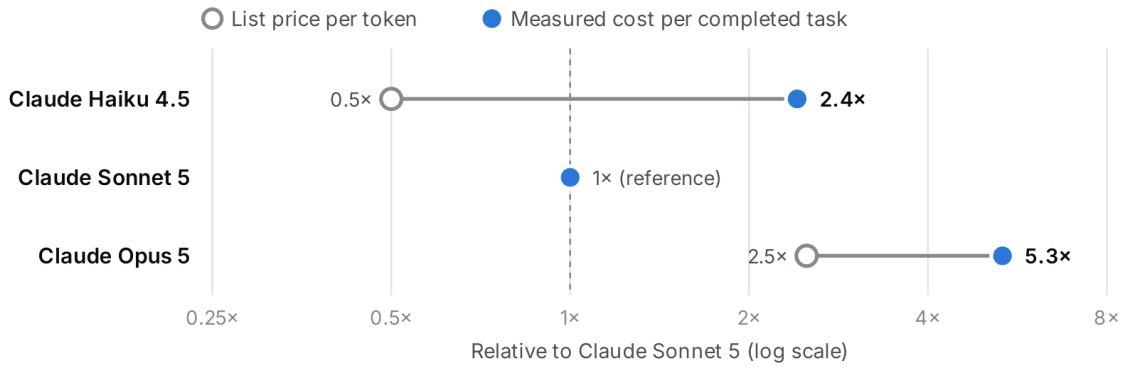

Figure 3. Price per token versus measured cost per completed task, relative to Sonnet 5. Haiku's half-price tokens became 2.4× the task cost. Opus's 2.5× price became 5.3×, because it also took twice Sonnet's turns at its default effort.

This also explains why the cascade policy (start on Haiku, escalate on failure) could not be tested properly. In smoke runs its deployable checker (visible tests still green, something in scope changed, nothing out of scope changed) accepted Haiku's partially correct work, so it never escalated. Escalating when Haiku hit its turn cap would have caught only 8 of 23 Haiku failures in calibration while needlessly escalating 15 of 109 passes. When escalation was forced with the hidden tests as checker, a second problem appeared: Sonnet, inheriting Haiku's partial edits, failed a task it passes reliably from a clean checkout. A wrong start anchors the next model.

5 Why routing helped less than expected

The disappointment is easiest to see in one number: **the hindsight oracle beat always-Sonnet by only 1.3%** ($0.0976 versus $0.0989). A router cannot save more than the oracle. On this workload, routing's entire upside was realized by picking one good default, and any router with imperfect judgment could only add cost. Four conditions produced that.

1. **No task needed the expensive model.** Where Haiku failed, Sonnet almost always passed; nothing required Opus. Routing earns money when the workload splits between tasks only the costly model can do and tasks the cheap model does cheaply. This one did not split.
2. **The cheap tier was not cheap per task.** A router's savings on easy work depend on the cheap model actually being cheaper on that work. Haiku 4.5 was not, even when it passed.
3. **The price ladder is narrow.** RouteLLM's 85% came from routing between GPT-4 Turbo and Mixtral 8x7B[1], a per-token gap of more than 20×. Claude's current ladder spans 5× from Haiku to Opus, and only 2× between adjacent tiers. Differences in how many tokens each model uses per task easily swamp gaps that small.
4. **The harness had already captured the easy wins.** Claude Code already uses Haiku for background work and exploration subagents, lets parents set subagent models, and offers effort levels and an Opus-plan/Sonnet-execute mode[21]. The author's own transcripts show parents routing subagents to Sonnet 9 times in 10. The obvious gain, not running every worker on Opus, is already partly in place.

6 Is it the models or the harness?

It helps to separate the three layers that set cost per task, because each is owned by someone different and each could change.

6.1 The price ladder (set by the vendor)

Per-token prices determine the maximum possible routing gain before behavior is counted. With Opus at 2.5× Sonnet and Sonnet at 2× Haiku, the ceiling is modest to begin with. Routing literature from 2023 and 2024 assumed ladders an order of magnitude wider. As frontier prices fall faster than small-model prices, this ceiling shrinks further; I could not find a paper that measures the effect directly, but the arithmetic is straightforward.

6.2 Tokens per task (set by training)

How many turns and tokens a model spends on a task is a trained behavior, and it does not track capability monotonically. Anthropic states that Opus 4.5 at medium effort matched Sonnet 4.5's best SWE-bench Verified score with 76% fewer output tokens[22], an example of a larger model being cheaper per task. In this study the ordering was different: Sonnet 5 was the most economical, using about half of Opus 5's turns (at its default effort) and a quarter of Haiku 4.5's. Haiku kept exploring after the job was done; Sonnet stopped. That is a property of how each model was trained to decide it is finished.

One confound matters: Haiku 4.5 is a generation older than the Sonnet 5 and Opus 5 models it was tested against. Part of its penalty may be generational rather than a property of the small tier. A Haiku-class model trained with the newer generation's habits could change the picture, and this is the first thing to re-test when one is available.

Could training remove the need for routers? The research trend points toward moving the routing decision inside the model:

- OpenAI's GPT-5 launched with a real-time router between fast and reasoning models, trained on signals such as users switching models and measured correctness, and OpenAI said it plans to integrate this into a single model[18].
- AdaptThink trains a model to choose whether to think at all, cutting response length by 40% to 53% on math while slightly raising accuracy[17].
- ARES uses a small controller to pick reasoning effort per agent step within one model, and reports up to 52.7% fewer reasoning tokens than fixed high effort with little loss in success on tool-use, research and web agents[16].

Choosing effort within one model keeps the same policy and, on some models, the same cache. Claude Code's documentation says changing effort on Opus 5.5 and Fable 5.1 keeps the prompt cache, while switching models never does[20]. I found no head-to-head study of effort control against cross-model routing on the same agentic benchmark, so the case for effort control rests on mechanism, not direct measurement.

6.3 The agent loop and caching (set by the harness)

The harness decides how context grows, how long caches live and where models can change. Three harness behaviors shaped these results:

- **Context re-read per turn.** Every turn re-reads the full conversation, which is what turns a 3.5× turn gap into a 7× read gap. Compaction and tighter tool outputs reduce the slope for every model, but they help most for models that take many turns.
- **Model-scoped caches.** Because caches are per model, routing is only cheap at boundaries where the cache is empty anyway. That is why the Claude Code team uses subagent handoffs rather than mid-conversation switches for Haiku work[19], and why GitHub says Copilot Auto "routes along natural cache boundaries"[27]. This study routed at exactly such a boundary, so its result is not a cache artifact.
- **Cache lifetime.** Claude Code writes caches with the 1-hour lifetime at 2× the input price, and cache writes were 44% of Sonnet's cost per task. Worker sessions here lasted from about 20 seconds to 4 minutes. At the 5-minute rate (1.25× input) the same writes would cost 37.5% less, which works out to roughly 16% off a Sonnet worker. That estimate is arithmetic on the recorded tokens, not a measurement, and it depends on whether the harness exposes that choice.

Harness designs with better evidence for routing in agent loops:

- **Explore, then route.** SWE-Router lets the cheap model work for a few turns before deciding whether to continue or escalate, and proves that conditioning on that partial trajectory never hurts the routing decision[13]. This targets the "difficulty is invisible up front" problem.
- **Hand off, don't switch.** A strong model writes a focused handoff for a cheaper worker (Claude Code's Explore agents[19], Aider's architect/editor split[24]). This study tested the model choice at such a handoff; it did not test whether a better brief lets a cheaper model succeed.
- **Cascades need a real verifier.** This study's own cascade never escalated because it had no reliable check. The literature's cascade wins (FrugalGPT[2], AutoMix[4]) rely on a scorer or self-verification that works for their task type.

ANSWER IN BRIEF

The weak routing payoff is not a bug in Claude's training or in Claude Code. It follows from a narrow price ladder, a mid-tier model that happens to be the most token-efficient on this work, and an agent loop in which extra turns cost more than linearly. Training could change the second (efficient small models, effort chosen inside the model), and harness design could change the third (explore-then-route, verified cascades, cheaper cache lifetimes for short workers). Neither would help much on a workload where one model already does everything well at the lowest price.

7 What the wider research shows

7.1 Headline router results come from single-turn benchmarks

Study	Reported result	Setting
FrugalGPT (2023)[2]	Matches GPT-4 with up to 98% cost reduction, or +4% accuracy at equal cost	Single-turn classification and QA, learned cascade
Hybrid LLM (ICLR 2024) [3]	Up to 40% fewer calls to the large model with no quality drop	Single-turn instruction following
AutoMix (2023)[4]	Over 50% cost reduction at comparable performance	Single-turn reasoning and QA with self-verification
RouteLLM (2024)[1]	At 95% of GPT-4 quality: over 85% savings on MT-Bench, 45% on MMLU, 35% on GSM8K	Chat, knowledge, math; GPT-4 Turbo vs Mixtral
MixLLM (2025)[6]	97.25% of GPT-4 quality at 24.18% of the cost	Single-turn
RouterEval (2025)[7]	Routers can beat the best single model, but most “still have significant room for improvement”	12 single-turn benchmarks, 8,500+ models

These savings are measured per query against a much more expensive model, with a lenient quality bar (for example 95% of GPT-4) and a price gap of 20× or more.

7.2 Independent re-evaluations are less kind

- **LLMRouterBench** (2026), 400,000+ instances across 33 models and 10 routers: “several recent approaches, including commercial routers, fail to reliably outperform a simple baseline,” with a large gap to the oracle driven by model-recall failures[8].
- **RouterArena** (2025), an open router leaderboard: GPT-5’s router reached 74.0% accuracy at $14.02 per 1,000 queries, Azure’s model router 68.1% at $0.54, Not Diamond 68.0% at $9.34, and the best academic router 66.9% at $0.15. No router led on every metric[10].
- **When Routing Collapses** (2026) reports that under generous budgets existing routers send nearly all queries to the strongest model, while an oracle would use it for under 20%[9].
- **Rerouting LLM Routers** shows that short appended text can force routers to always pick the expensive model[11].

7.3 Agentic and coding studies

- **The Replay Gap** (CMU, August 2026): swapping models mid-trajectory changed 61% to 94% of later actions, and the common offline method of evaluating per-step routers on stitched logs mispredicted every outcome that mattered. Router savings estimated offline from agent logs are therefore unreliable[12].
- **SWE-Router** (June 2026): prompt-only routing for software tasks has an inherent error floor because difficulty is hidden in the repository; letting the cheap model explore first improves routing decisions[13].
- **Applied Compute** (2026) trained a router over Nemotron 3 Ultra, Claude Opus 4.7 and GPT-5.5 on SWE-bench Verified. Always-Opus passed 83.4%, the router about 76% (roughly GPT-5.5’s quality at 25% lower cost than GPT-5.5), and a perfect oracle 89%[14]. The router traded several points of completion for its saving.

- **Scout-then-fix** (2026, as reported): on SWE-bench Pro, a cheap scout that localizes the bug and hands off to a fixer matched Opus 4.6's solve count at under a fifth of the cost per solve, but a no-router ablation scored the same, so the saving came from the handoff design, not the routing decision[15].
- **Aider**'s architect mode (a strong model plans, another edits) produced mainly quality gains, for example R1 plus Sonnet at 64.0% on the polyglot benchmark against Sonnet alone at 51.6% for similar cost[24].

The pattern across these is consistent with this study. Choosing among models once per task can pay when tiers genuinely differ in which tasks they can solve. Switching models within a trajectory is hard to evaluate and hard to profit from. Much of the reported gain comes from how the work is split and handed off rather than from the routing classifier.

7.4 Commercial tools: claims versus independent measurement

Tool	Vendor claim	Independent evidence
AWS Bedrock Intelligent Prompt Routing[25]	"Reduce costs by up to 30% without compromising on accuracy"; routes within one model family	NONE RIGOROUS Only anecdotal practitioner tests found
Azure / Foundry model router[26]	Balanced, Cost and Quality modes with stated quality bands	MEASURED RouterArena: 68.1% at $0.54 per 1k queries, the best cost efficiency among commercial routers[10]
GitHub Copilot Auto[27]	10% discount on model costs; routes along cache boundaries; "improved cost efficiency"	NONE FOUND No published numbers
OpenRouter Auto Router[28]	Routes by task type using community usage; vendor MMLU-Pro result of 85.2% at about a third of its previous router's cost	NONE FOUND
Not Diamond	Savings claims appear only in secondhand sources	MEASURED RouterArena: 68.0% at $9.34 per 1k queries[10]
Cursor Auto[29]	Cost, Balance and Intelligence modes; bills at the list price of the chosen model	NONE FOUND
LiteLLM auto router[30]	Own benchmark: about 58% savings on 300 prompts, 46.8% win rate against the flagship	SELF-REPORTED

The only independent multi-vendor measurements found (RouterArena, LLMRouterBench) use single-turn QA. I found no independent study of real-world savings from Copilot Auto, Cursor Auto, Windsurf or OpenRouter Auto on agentic coding.

Two practical observations follow. Several of these tools are priced so that the platform benefits from routing even when the user's quality-adjusted cost does not improve (a discount for using Auto, a flat rate against a quota). And none publish cost per completed task on agentic work, which is the number this study suggests is decisive.

8 When routing pays and when it does not

Routing tends to pay when

- The **price gap per task** between tiers is wide, not just the gap per token.

- A large share of traffic is **clearly easy and recognizable from the request**, as with FAQ or classification traffic.
- Some work **genuinely needs the expensive model** and some does not, so the oracle beats every fixed default by a real margin.
- Requests are **short or start fresh**, so there is little cache to lose.
- The **quality bar tolerates** occasional weaker answers.
- A **reliable verifier** exists, so cascades can escalate on real failures.

Routing tends not to pay when

- It **switches models inside a long, cache-warm conversation**.
- The cheap model **takes more turns or fails and retries**, as Haiku did here.
- One model is already the **cheapest per completed task** across the workload.
- The harness **already routes the easy wins** (background tasks, exploration subagents).
- Routers are **evaluated offline on logs** or against the most expensive model instead of the best fixed default.
- The task's difficulty is **only visible after exploration**.

A test before buying or building a router: run each candidate model alone on a representative sample of real tasks, graded by outcome, and compute the oracle. If the oracle is not clearly cheaper than the best single model, no router will be either.

9 Limitations

- **Synthetic, well-specified tasks.** Three small Python repositories with short, clearly scoped briefs. Harder, vaguer or longer work may open headroom (a task only Opus can do) or remove it (Sonnet may start failing). The results speak to agentic coding handoffs, not chat or knowledge work.
- **No Opus-only tasks.** The calibration found none, so the study cannot show the case where routing to a frontier model is worth its price.
- **Generation mismatch.** Haiku 4.5 was compared with Sonnet 5 and Opus 5. Model aliases also move over time; the results describe the models as served on 24 and 25 September 2026.
- **Effort not varied in the static arms.** Opus ran at its default effort. "Opus at low effort" was on the router's menu but was never tested as a fixed default, and Sonnet's effort was not varied.
- **Brief quality held constant.** The worker always received the same canned brief, so the study isolates the model choice and does not measure whether a parent writes better briefs for cheaper models.
- **Single vendor, single operator, list prices.** One model family and one harness. Price ratios differ across vendors, so the Sonnet-versus-router gap in particular may not carry over. Subscription allowance consumption was not measured directly.

- **The key practical comparison was exploratory.** The always-Sonnet result was not pre-registered. It is reported with intervals and should be confirmed in a registered follow-up.
- **Literature review caveats.** Several 2026 papers were read from abstracts or HTML summaries; figures marked “as reported” were not independently re-derived.

10 Next experiments

1. **Register always-Sonnet as the control** and test the router against it, so the practical claim becomes confirmatory.
2. **Add headroom above Sonnet**: longer, vaguer, multi-module tasks until some are Opus-only in calibration, plus a long-brief stratum matching real subagent dispatches (1,000 to 2,500 tokens).
3. **Make effort a routing dimension**: static arms for Sonnet and Opus at low and medium effort, and an effort-only router on one model, to test in-model adaptive compute against cross-model routing.
4. **Explore-then-route**: start on the cheap model for a few turns, then let it or the parent decide whether to continue or escalate in a clean session.
5. **Give the cascade a real verifier** (generated tests or a reviewer agent) and restart escalations from a clean checkout to avoid anchoring on the cheap model’s partial work.
6. **Replicate on another vendor’s ladder** (the harness already has a Codex track) and re-run Haiku when a newer-generation small model is available.
7. **Measure cache lifetime**: compare 1-hour and 5-minute cache writes for short worker sessions where the harness allows it.

Appendix A. Reproducibility

Item	Value
Run	results/exp05_dispatch/20260924-125257, marked confirmatory
Registered at	2026-09-24 16:52 UTC, git aa0d54cd17fd
Task set hash	5b99585211dedaaf… (65 tasks)
Config hash	261b1cc76a9dd412…
Design	Paired; 3 trials; seeded randomized blocks (seed 20260923); sequential sessions; 40-turn cap
Statistics	Task-clustered bootstrap intervals; two-sided task-clustered permutation tests; one-sided bootstrap non-inferiority test at 10 points; Holm adjustment across secondary hypotheses
Spend	$202.33 list price for the confirmatory run, including $16.34 of parent-seeding setup excluded from headline costs; $109.60 for calibration
Environment	Claude Code CLI 2.1.278; Python 3.14.7; macOS 26.6.2 (arm64)
Clock time	About 16 hours, with one planned stop and resume at cell 336 to raise the budget; no usage limits or provider outages occurred

Rebuild the statistics with `make dispatch-report RUN=<run_dir>`. The per-task pass/fail matrix for all 780 cells is in the repository's paper draft.

Appendix B. Earlier single-turn experiments

Before the agentic study, the same harness tested routing on single-turn knowledge work (classification, extraction, arithmetic over tables, handbook questions). The samples were small (5 and 8 tasks) and every policy passed every task, so they show cost shape only. Two observations carried into the agentic design: a Haiku classifier used as the router spent 500 to 900 thinking tokens per decision, making routing overhead 63% of that policy's spend; and a simple keyword heuristic router kept an 87% cache hit rate and was the cheapest policy, while the oracle, spreading work across three models, managed only 57%.

References

1. Ong et al. RouteLLM: Learning to Route LLMs with Preference Data. 2024. arxiv.org/abs/2406.18665; lmsys.org/blog/2024-07-01-routellm
2. Chen, Zaharia, Zou. FrugalGPT. 2023. arxiv.org/abs/2305.05176
3. Ding et al. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. ICLR 2024. arxiv.org/abs/2404.14618
4. Aggarwal et al. AutoMix: Automatically Mixing Language Models. 2023. arxiv.org/abs/2310.12963
5. Hu et al. RouterBench. 2024. arxiv.org/abs/2403.12031
6. Wang et al. MixLLM. NAACL 2025. arxiv.org/abs/2502.18482
7. Huang et al. RouterEval. EMNLP Findings 2025. arxiv.org/abs/2503.10657
8. LLMRouterBench: A Massive Benchmark and Unified Framework for LLM Routing. 2026. arxiv.org/abs/2601.07206
9. When Routing Collapses. 2026. arxiv.org/abs/2602.03478 (as reported)
10. RouterArena: An Open Platform for Comprehensive Comparison of LLM Routers. 2025. arxiv.org/abs/2510.00202
11. Shafran et al. Rerouting LLM Routers. 2025. arxiv.org/abs/2501.01818
12. Gonuguntla. The Replay Gap: Static Evaluation of Model Switching in LLM Agents Scores the Wrong World. CMU, August 2026. arxiv.org/abs/2608.08239
13. Son et al. SWE-Router: Routing in Multi-turn Agentic Software Engineering Tasks. June 2026. arxiv.org/abs/2607.00053
14. Applied Compute. Training an Agentic Router for Software Engineering Tasks. 2026. appliedcompute.com/research/training-an-agentic-router
15. Scrouting / SuperScout. 2026. arxiv.org/abs/2608.04804 (as reported)

16. ARES: Adaptive Reasoning Effort Selection for Efficient LLM Agents. March 2026. arxiv.org/abs/2603.07915
17. Zhang et al. AdaptThink: Reasoning Models Can Learn When to Think. 2025. arxiv.org/abs/2505.13417
18. OpenAI. GPT-5 System Card. 2025. openai.com/index/gpt-5-system-card
19. Shihipar. Lessons from building Claude Code: prompt caching is everything. Anthropic, April 30, 2026. claude.com/blog/lessons-from-building-claude-code-prompt-caching-is-everything
20. Anthropic. Claude Code documentation: prompt caching. code.claude.com/docs/en/prompt-caching
21. Anthropic. Claude Code documentation: model configuration. code.claude.com/docs/en/model-config
22. Anthropic. Introducing Claude Opus 4.5. anthropic.com/news/claude-opus-4-5
23. Anthropic. Introducing Claude Haiku 4.5. anthropic.com/news/claude-haiku-4-5
24. Aider. Separating code reasoning and editing (architect mode), 2024; R1 + Sonnet results, 2025. aider.chat/2024/09/26/architect.html; aider.chat/2025/01/24/r1-sonnet.html
25. AWS. Amazon Bedrock Intelligent Prompt Routing. aws.amazon.com/bedrock/intelligent-prompt-routing
26. Microsoft. Model router concepts. learn.microsoft.com/en-us/azure/foundry/openai/concepts/model-router
27. GitHub. About Copilot auto model selection. docs.github.com/copilot/concepts/auto-model-selection
28. OpenRouter. Introducing the new Auto Router. August 2026. openrouter.ai/blog/announcements/introducing-the-new-auto-router
29. Cursor. Models and pricing. cursor.com/docs/models-and-pricing
30. LiteLLM. Auto routing benchmark. docs.litellm.ai/docs/proxy/auto_routing_benchmark