# ATTEMPT-MATCHED EVALUATION OF MODEL-ROUTING CASCADES IN LLM AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A model-routing cascade tries a cheap model first and falls back to a stronger model after failure. Comparing this policy with a single strong-model attempt confounds model choice with the benefit of trying again. We isolate model choice by comparing the cascade with a same-model retry at the same two-attempt budget. The result differs across two agent benchmark families: on $\tau^2$-bench, the routing effect is indistinguishable from zero on most datasets and significantly negative on one; on BFCL's multi-turn split, it is significantly positive. An exact identity separates the cost of using a weaker model first from the benefit of recovering failures. The strong model's *self-rescue rate*, its success rate when retried after its own failure, correlates with the routing effect across 11 cells (rank correlation $-0.84$). This association is established between benchmark families, with shared tasks across cells; it does not identify a cause within either family. Retry accounts for up to 6.0 times the naive cascade gain (resampled lower bound 2.6). Cascading still reduces cost on $\tau^2$; on BFCL, the evaluated pair improves both quality and cost over retrying its stronger member, and the pool's best fixed two-slot policy is a cascade. These results concern reactive cascades. A one-call dispatcher's selection-corrected two-arm oracle excludes zero on 2 of 5 datasets. Our retries are blind and our results depend on the serving path; a retry that benefits from failure feedback could further reduce the measured routing effect.

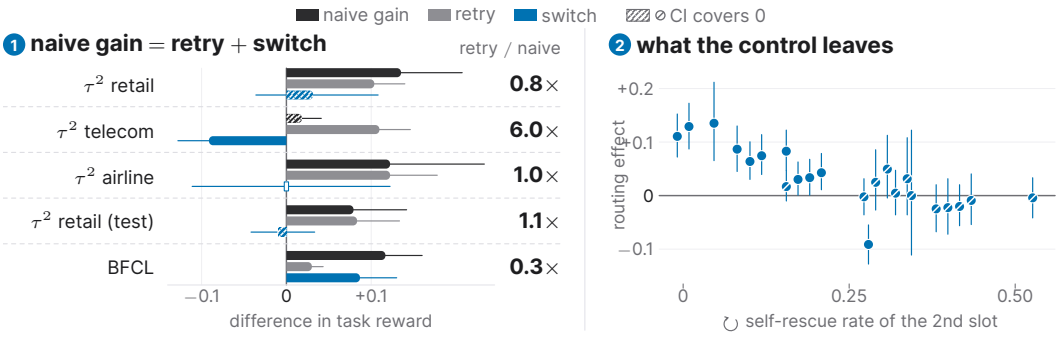



Figure 1: Cascade gains and the retry control. 1 Each dataset has three differences with task-bootstrap 95% CIs: *naive gain* and *retry effect* use one-slot always-strong as the baseline; *switch* (the routing effect) uses the two-slot retry policy. Solid marks have intervals excluding zero; striped marks cover zero, with outlines used for narrow marks. The right column reports retry's share of the naive gain, 0.3 to 6.0 (intervals in Table 29). Two marks are not drawn to scale: `airline`'s switch term is exactly 0, shown as an open tick, and `retail` test's uses the minimum bar length. 2 Routing effect versus the second-slot model's self-rescue rate for 23 measurements across 11 (dataset, second-slot model) cells. The cell-level rank correlation is $-0.84$. Uncorrected intervals exclude zero from above for 10 measurements and from below for 1; 12 cover zero. Tied rates are offset for visibility. Section 5 reports multiplicity corrections.

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.

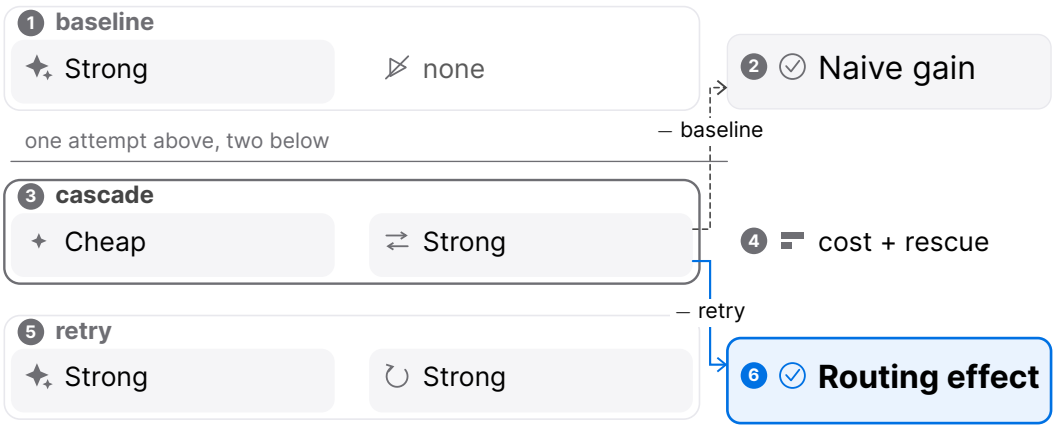



Figure 2: Separating model choice from the attempt budget. Each row is a policy; columns show its first and second attempts. ❶ Always-strong uses one attempt. ❷ The naive cascade gain compares policies with different attempt budgets (dashed line). ❸ The cascade tries the cheap model, then a different model after a verifier-detected failure. ❹ The identity in Section 4 decomposes the routing effect. ❺ Retry uses a separate rollout of the same strong model; endpoint nondeterminism can make retrying useful even at temperature 0. ❻ Cascade minus retry is the routing effect at a matched two-slot budget (solid line).

## 1 Introduction

Model routing in LLM agents is often motivated by a per-task oracle: run every task with every candidate model, select the best outcome, and compare it with the best fixed model (Zhou et al., 2026b). A related comparison evaluates a cheap-then-strong cascade against one strong-model attempt. On $\tau^2$-bench's `retail` split, that cascade gains $+0.14$, the largest naive gain in Table 1.

This comparison combines two benefits: choosing a different model and making another attempt. A reactive cascade (Chen et al., 2023a; Jitkrittum et al., 2023) calls a second model after failure, whereas always-strong gets only one attempt. Even at temperature 0, repeated rollouts can differ (Figure 2). To measure the contribution of model choice, we therefore compare the cascade with a retry of the same strong model.

The attempt mismatch applies to reactive fallback cascades. A *one-call* dispatcher commits to one model before execution and has no such mismatch. Our two-arm one-call oracle retains 25% to 82% of the two-slot naive gain after correcting for arm selection, with intervals excluding zero on 2 of 5 datasets. Across the full 5-model pool, the selection correction outweighs the benefit of additional arms (Appendix A.14). Our negative results below are therefore specific to cascades.

The retry control gives different answers on two independently implemented benchmark families (Figure 1). On $\tau^2$-bench, retry accounts for most or all of the naive gain. The residual is indistinguishable from zero on most datasets and significantly negative on one. On BFCL's multi-turn split (Mao et al., 2026; Patil et al., 2023), the residual is significantly positive.

An exact identity separates this residual into a first-slot cost and a rescue term. Starting with a weaker model reduces first-slot success; switching models can recover failures that a retry would miss. The second term favours the cascade in our measurements, but its net contribution depends on the strong model's *self-rescue rate*: how often a second rollout succeeds after the first fails. A high self-rescue rate strengthens the retry baseline. A large capability gap can also make the first-slot cost dominate, as on `telecom`.

The choice of baseline remains important after attempts are matched. On BFCL, 9 of 10 pairs improve significantly over retrying their own better member before correction (5 after correction; Section 5). No $\tau^2$ pair is shown to beat the best fixed two-slot policy in the full pool. On BFCL, the best such policy is itself a cascade; no other pair is shown to beat it.

Our contributions are the retry-controlled comparison on 4 $\tau^2$ datasets and BFCL (Section 3), the exact decomposition and its cross-family analysis (Sections 4–5), and an evaluation of practical alternatives. The lexical router and transcript judge we tested recover none of the measured gain, although these results leave semantic routers and improved judges open (Section 6). We also measure the support available for offline evaluation: both session-level and step-level estimators are unbiased under their matched logging designs, but neither is reliable from one log at our sample sizes, and step-level support falls exponentially with horizon (Section 7). Appendix B reports per-arm noise floors and the condition under which trial noise inflates a per-task-maximum oracle.

## 2 SETUP

**Benchmarks.** We use $\tau^2$-bench (Barres et al., 2025) at a pinned commit, with `retail`, `telecom` and `airline` domains. It extends the tool-agent-user setting of Yao et al. (2024) by requiring the agent to coordinate actions with a user. Reward combines the benchmark's database-state comparison and rule assertions. We fix the user simulator across arms and use the strong-arm model as the natural-language assertion grader. Fixing the grader removes between-arm variation but leaves a risk of self-preference. Episodes are capped at 50 steps, or 100 on `telecom`; 28 of 2,196 $\tau^2$ rollouts reach a cap. BFCL has no step cap (Appendices A.5 and A.1). Main experiments use the training splits. We report the held-out `retail` test split without using it to select hyperparameters, configurations or thresholds.

**Models and repetition.** The primary pair uses one family's cheap and strong tiers, tagged `haiku45` and `sonnet45`. The pool adds `gpt56`, `glm5` and `qwen3`, giving 5 models from 4 vendors. We use `opus45` only as a judge (Section 6; endpoints in Table 2). Temperature is 0 where supported; `gpt56` rejects this setting and runs at the provider default. BFCL's primary pass uses a custom handler to pin temperature. An earlier unpinned pass is labelled secondary throughout; changing the handler also changed the API surface (Section 5, Appendix A.2). We run each task three times per arm to measure trial variability and supply independent rollout records for the retry control. The trial index never reaches the endpoint, so differences reflect endpoint nondeterminism rather than controlled seeds. The $\tau^2$ null results are conditional on this serving path (Appendix A.26).

**Attempt matching and cost.** We score each two-attempt policy over all ordered trial pairs $(i, j)$ with $i \neq j$, avoiding self-comparisons (Appendix A.4). We match the number of available *slots*. The verifier triggers the second slot only after failure, so mean call counts still differ by the capability gap. With binary reward, this difference affects cost but not reward accounting (Appendix A.26). Cost includes the first attempt and any triggered second attempt. Each attempt is a complete episode from the benchmark's initial state. Both harnesses support this reset; a deployment with non-transactional tool effects may not (Chen et al., 2026b).

## 3 SEPARATING RETRY FROM MODEL CHOICE

Let $r_{t,k,i}$ be the reward of task $t$ under model $k$ on trial $i$; a two-slot policy $k_1 \to k_2$ runs $k_1$ first and falls back to $k_2$ when $r < 1$. With $c$ the cheap model and $s$ the strong one,

$$\underbrace{\mathrm{V}(c \to s) - \mathrm{V}(s)}_{\text{naive gain, what is usually reported}} = \underbrace{\mathrm{V}(s \to s) - \mathrm{V}(s)}_{\text{retry effect}} + \underbrace{\mathrm{V}(c \to s) - \mathrm{V}(s \to s)}_{\text{routing effect}}.$$

All policies use the benchmark verifier for escalation, so only model assignment changes. Cascade work reports the leftmost quantity (Chen et al., 2023a). A perfect trigger gives routing its most favourable setting in this design: we ask whether changing models outperforms trying the same model again. With binary reward and symmetric ordered trial pairs, the two cascade directions have exactly the same reward. Thus this quantity measures the benefit of covering tasks with a model pair, before learning any routing rule (Appendices A.26 and A.9).

The retry effect is consistently large in Table 1. The routing effect's interval covers zero on `retail`, `airline` and the held-out `retail` split. These intervals differ in width, so the claim that retry accounts for the gain describes the point estimates rather than establishing equivalence. On `telecom`,

| | | one-slot | two-slot decomposition | | | |
|---|---|---|---|---|---|---|
| benchmark / dataset | $T$ | strong | naive | retry | **routing** | routing 95% CI |
| $\tau^2$ retail train | 74 | 0.685 | $+0.135$ | $+0.104$ | $+0.032$ | $[-0.036, +0.108]$ |
| $\tau^2$ telecom train | 73 | 0.607 | $+0.018$ | $+0.110$ | $-0.091^*$ | $[-0.128, -0.055]$ |
| $\tau^2$ airline train | 30 | 0.644 | $+0.122$ | $+0.122$ | $+0.000$ | $[-0.111, +0.122]$ |
| $\tau^2$ retail test | 40 | 0.842 | $+0.079$ | $+0.083$ | $-0.004$ | $[-0.042, +0.033]$ |
| BFCL `multi_turn` | 200 | 0.697 | $+0.117$ | $+0.030$ | $+0.087^*$ | $[+0.045, +0.130]$ |
| *unpinned* $T$ | 200 | 0.702 | $+0.095$ | $+0.048$ | $+0.047^*$ | $[+0.015, +0.081]$ |

Table 1: Decomposition of oracle cascade gain on two benchmark families. The naive gain compares the cheap-then-strong cascade with one-slot always-strong; $^*$ marks a routing effect whose 95% CI excludes zero. Intervals use 10,000 paired task-bootstrap samples with the three rollouts per cell fixed. Trial resampling changes no significance conclusion in these rows (Appendix A.13); Table 29 gives intervals for all terms. BFCL's pinned pass is primary; the earlier unpinned pass could not fix temperature (Appendix A.2). Cost appears in Figure 3b. $T$ counts tasks. `telecom` retains 73 of 74 launched tasks after one infrastructure error removes a task from the reward analysis (Appendix A.1).

the routing effect is negative under both task-only and hierarchical bootstrapping (Appendix A.13). Its large capability gap explains this result through the first-slot cost (Section 4). With three slots, the `retail` effect grows to $+0.0586$ (one-sided $p = 0.028$), but it does not remain significant after correction across the four datasets (Appendix A.15).

The $\tau^2$ results therefore show no reliable positive residual after accounting for retry. They do not tightly bound every residual: interval upper limits reach 80% of `retail`'s naive gain and 100% of `airline`'s. Of the 4 intervals, 1 cover zero informatively, 2 are too wide to do so, and 1 is significantly negative (Appendix A.19).

These results concern two-slot policies. The selection-corrected one-call oracle over the same two models still gains $+0.1014$ $[+0.0248, +0.1509]$ on `retail`, compared with $+0.1306$ when selection and scoring reuse rollouts (Appendix A.14). The retry control changes how we interpret the cascade gain; it does not eliminate one-call headroom. The deployable alternatives in Section 6 fail to recover that headroom.

## 4 An Exact Identity for the Routing Effect

A near-zero residual could arise because the models fail on the same tasks, or because the benefit of recovering different failures offsets the cost of starting with a weaker model. Binary reward lets us separate first-slot success from conditional rescue exactly:

$$\underbrace{\mathrm{V}(c \to s) - \mathrm{V}(s \to s)}_{\text{routing effect}} = \underbrace{(p_c - p_s)}_{\text{first-slot cost} \le 0} + \underbrace{(1 - p_c)\,\mathbb{E}[s \mid c \text{ fails}] - (1 - p_s)\,\mathbb{E}[s \mid s \text{ fails}]}_{\text{rescue term}},$$

where $p_k$ is the one-slot success rate; the conditional expectations run over the same ordered trial pairs, so the rescuing attempt is always a separate rollout. No independence assumption enters, which matters because the retry policy's two slots land on the same task and are strongly correlated (Appendix A.8 derives it and bounds its reach).

Table 11 reports the difference between the two conditional rescue rates, which we call the complementarity gap. It is positive on all 4 $\tau^2$ datasets, significantly so on 2, and ranges from $+0.0937$ to $+0.3267$ across both families. This descriptive gap is several times the routing effect, but it does not establish causal complementarity: the two failure events select tasks of different difficulty. Using the second-slot model's unconditional accuracy as a reference reverses the comparison on all 23 of 23 measurements, with differences from $-0.087$ to $-0.337$. Neither reference matches difficulty. We did not run a matched-difficulty experiment or an additional arm to resolve this ambiguity.

The rescue term offsets 76% to 499% of the first-slot cost across our measurements. A larger capability gap, $p_s - p_c$, therefore requires a larger rescue term to make routing beneficial. On `telecom`, that term is too small to compensate for spending the first attempt on the weaker model.

**Self-rescue and the retry baseline.** The identity subtracts $(1 - p_s)\mathbb{E}[s \mid s \text{ fails}]$, the gain from retry. We call the conditional rate $\mathbb{E}[s \mid s \text{ fails}]$ the *self-rescue rate*. It measures how often a separate rollout succeeds after the first fails and depends on both model and evaluation conditions. A higher rate strengthens the retry baseline. Its point estimate varies $1.9\times$ across the four $\tau^2$ datasets (Appendix C), but these dataset-level intervals overlap pairwise. Their widths range from $0.000$ to $0.561$ over 23 measurements (Table 16). BFCL's interval is disjoint from all four, so the clearest contrast is between benchmark families (Section 5). Estimating these rates on a small profiling sample does not yield a reliable sign predictor (Appendix A.16).

## 5 A Second Benchmark Family Reverses the Sign

The $\tau^2$ domains share a harness, grader design and language-model user simulator. To test whether the result extends beyond this implementation, we repeat the decomposition on BFCL's `multi_turn_base` split. BFCL grades post-execution object state and uses scripted user messages, without a language-model user simulator.

On BFCL, the routing effect is significantly positive: $+0.0867$ $[+0.0450, +0.1300]$ (Table 1). Retry accounts for 26% of the naive gain. The self-rescue rate is $0.0989$, compared with $0.2791$–$0.5263$ on $\tau^2$. The cross-model rescue rate, $0.4256$, lies just below $\tau^2$'s $0.4615$–$0.6200$ range. The conditional rescue gap thus persists across families, while the benefit already available from retry decreases.

The primary BFCL pass pins temperature at $0$. An earlier pass at the provider default gives a routing effect of $+0.0467$ $[+0.0150, +0.0808]$. The pinned pass has a lower self-rescue rate, $0.0989$ versus $0.1620$, and a larger routing effect, consistent with the identity. This is not a controlled temperature comparison because the API path also changed. Both passes retain a positive effect; the secondary pass uses the harness's own handler (Appendix A.2).

The source of the cross-family difference remains unresolved. Self-rescue falls for all $5$ models tested in both families, but simulator, grader, task distribution, client and harness all change together. We did not run $\tau^2$'s solo mode, which removes the user simulator while also changing other conditions (Appendices A.13 and C). The results establish a cross-family association rather than identifying the simulator as its cause.

**A wider model pool.** We expand the `retail` pool to $5$ models from $4$ vendors and evaluate all $10$ pairs (Table 17). One pair has a significantly positive routing effect relative to its own retry baseline. Its second-slot model has self-rescue $0.0465$ and cross-rescue $0.4747$. Relative to the best two-slot policy in the full pool, 0 of the $10$ pairs is shown to improve. This does not establish strict domination, since the pool best is selected in sample and the comparisons are uncertain (Appendix A.20), consistent with the evaluation concerns in Zhou et al. (2026a).

**The BFCL pool.** Table 15 repeats the analysis on BFCL. Relative to retrying the member with higher one-slot accuracy, 9 of $10$ pairs improve significantly. Selecting the retried member by two-slot reward gives 9 significant pairs; resampling trials gives 7. Against the pool-best policy, only $1$ pair attains the best value, and it is the best policy itself (`cascade(sonnet45→glm5)`). Its margin against itself is exactly $0$.

No other BFCL pair is shown to exceed the pool best. What differs from the full $\tau^2$ pool is the type of best policy: a cascade on BFCL and a retry on $\tau^2$. This depends on decoding conditions. Restricting $\tau^2$ to the $4$ pinned arms selects `cascade(sonnet45→glm5)`, with a margin of $+0.0405$ $[-0.0113, +0.0991]$ that covers zero. Thus pinning changes the selected policy type without establishing a significant advantage (Appendices A.20 and A.1).

**Cross-family association and its limits.** The 23 measurements span $11$ (dataset, second-slot model) cells. At this level, routing effect and self-rescue have rank correlation $-0.84$ (permutation $p = 0.001$; Table 16). This association has three limitations (Appendix A.6). First, self-rescue

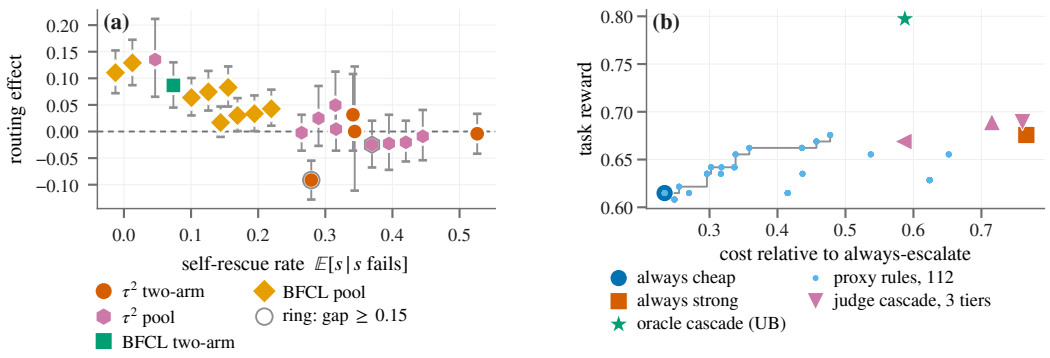



Figure 3: **(a)** Routing effect versus second-slot self-rescue across both benchmark families and the 5-model pool. Bars are task-bootstrap 95% CIs; tied rates are offset for visibility. The 11 cells have rank correlation $-0.84$. Grey rings mark capability gaps at or above 0.15; all measurements remain included. **(b)** Reward versus cost on `retail`. The grey line is the frontier of 112 proxy rules. Judge tiers are ▼ `haiku45`, ▲ `sonnet45` and ◀ `opus45`. Proxy rules reach at most always-strong's 0.6757; 2 judge cascades exceed it, reaching 0.6892, below the oracle's 0.7973. The plotted gains use one-slot always-strong. Under the main text's ordered-pair convention, two-slot retry reaches 0.8108 and the matched cascade 0.8041; 0 judge tiers reach either value. The oracle and deployable policies use different gates, detailed in Appendix A.6.

enters the routing identity with coefficient $-(1 - p_s)$. Removing that term leaves correlation $-0.33$ ($p = 0.16$). An identity-preserving permutation gives $p = 0.11$ against a null centred on $-0.64$, so the observed association does not establish a causal mechanism. Second, it is supported across families but not within either: the correlations are $-0.43$ within $\tau^2$, $-0.80$ within BFCL, and $-0.40$ over 5 datasets. Third, the capability gap gives a comparable ordering. Within $\tau^2$ its correlation is $-0.61$, compared with $-0.43$ for self-rescue. The paired difference in $|\rho|$ is $+0.14$ $[-0.48, +0.80]$, which does not distinguish the two at this cell count. Self-rescue is useful because it requires one extra rollout per task and no second model, but the descriptive boundary in Section 8 needs independent validation.

We correct the routing comparisons over 23 measurements, deduplicated by arm identity. 8 survive Bonferroni and 9 survive Benjamini–Hochberg; among the 9 BFCL pool pairs in this family, the corresponding counts are 5 and 6 (Appendix A.13).

## 6 Evaluation of Deployable Alternatives

**Lexical task-text routing.** The router sees the scenario text available before execution, without evaluation criteria. We select one of 11 feature/model variants using the training split, with further margin and threshold sweeps detailed in Appendix A.17. On held-out $\tau^2$ tasks, its gain is $-0.0250$ $[-0.1000, +0.0000]$ against selection-corrected oracle headroom of $+0.0500$, capturing $-50\%$. The interval is too wide to rule out a useful lexical router. We therefore repeat the evaluation on BFCL, with 199 tasks and 5 arms. The gain is $-0.0201$ $[-0.0553, +0.0000]$ against headroom of $+0.1307$, with the same sign and an interval $1.8\times$ narrower. These results concern count and $n$-gram features on two corpora; they leave semantic routing open (Appendices A.14 and A.17).

**Transcript-based failure detection.** A deployed cascade may lack the benchmark verifier. We replace it with one model call that reads the transcript and judges completion, using 3 tiers from one vendor (Appendix A.12). Model judges can have systematic biases (Lai et al., 2026). Here, detector accuracy does not consistently improve with tier, and all 3 pairs of accuracy intervals overlap across 148 judgements (Figure 4a). This sample does not establish an ordering. The Chinese prompt over English transcripts, twenty-character reason limit and arm-asymmetric truncation could affect performance. Moreover, 56 judgements use transcripts identical to another's; intervals cluster by task to account for this repetition. We did not ablate these choices (Appendix A.7).

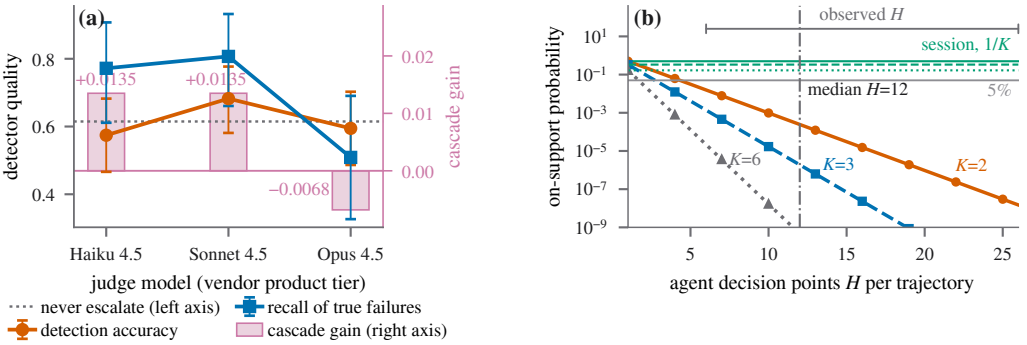



Figure 4: Failure detection and offline support. **(a)** Accuracy (circles) and failure recall (squares), with task-clustered 95% intervals. All 3 accuracy-interval pairs overlap, leaving the ordering by vendor tier unresolved. Bars use the right axis for cascade gain over one-slot always-strong; all intervals cover zero. The dotted line is never-escalate accuracy, 0.6149. **(b)** Support probability versus the number of agent decisions $H$. Step-level targets under $\beta_{\text{step}}$ have support $K^{-H}$; session-level targets under $\beta_{\text{sess}}$ have support $1/K$. The bracket shows the observed horizon range on `retail` train and the dash-dotted line its median.

Against one-slot always-strong, the best judge cascade captures 11% of the naive gain, with an interval covering zero. None of the 3 reaches the two-slot retry control (Figure 3b). These results quantify the loss for this particular escalation trigger. They do not isolate the routing effect under a shared deployable trigger, which would require a judge-gated retry arm that we did not run.

## 7 Logging Granularity and Offline Evaluation

Offline evaluation of a deterministic router requires logged data that support its decisions (Liu et al., 2026a; Gonuguntla, 2026). Consider two logging designs. Under $\beta_{\text{step}}$, a model is drawn uniformly from $K$ choices at every decision point. With an episode-end reward, a trajectory contributes to per-decision importance weighting (Precup et al., 2000) only when all $H$ choices match the target. This has probability $K^{-H}$; the agent's execution model determines $H$ (Nie et al., 2026). Under $\beta_{\text{sess}}$, one model is drawn and retained for the session, giving a session-level target support probability $K^{-1}$. A session-level logger has zero support for a step-level target that switches models mid-episode. Executing candidate policies would instead evaluate their deployed values (Yang et al., 2026a). The variance problem for long-horizon importance weighting is established (Liu et al., 2018; Jiang & Li, 2016); here we measure the horizons and the behaviour of the session-level estimator.

Figure 4b plots the support curves over `retail` train's observed horizons. Across 1,307 trajectories, the minimum horizon is 3. Only 6 trajectories have at least 5% step-level support at $K = 2$, all in `airline`. The pooled step-to-session ratio is 312×, equivalent to 625 logs per admissible trajectory. Domain-specific ratios range from 69× to 992×; 1 of the four falls below the recorded 100× criterion (Appendix A.11).

Because $K^{-H} > 0$ for finite $H$, the step-level estimator remains unbiased. Its practical limitation is the small number of trajectories with nonzero weight; an episode-end reward provides no intermediate observations to stabilise the estimate. Session-level evaluation is also unbiased, but it ranks policies unreliably from a single log at these sample sizes: on `retail`, the median per-log correlation with true policy ranks is 0.486 (Appendix A.18).

## 8 Cost and Reporting Recommendations

Cascading reduces cost on $\tau^2$, although it has no established quality advantage over retry. The best cascade among the 4 pinned arms also lacks a significant quality advantage (Section 5). At the same two-slot budget, retrying the strong model costs 1.18× to 2.08× as much as cheap-first (Table 31).

Quality intervals cover zero difference on three of the four $\tau^2$ datasets; on `telecom`, the cascade is significantly worse ($-0.0913$), consistent with its large capability gap.

On BFCL, the evaluated cascade improves quality from $0.7267$ to $0.8133$ relative to retrying its own stronger member, while that retry costs $1.95\times$ as much. Across the full pool, 1 of 10 pairs attains the best fixed two-slot value, and it is the pool-best cascade itself. The pool study records no cost, so the joint quality-and-cost improvement is established only for the primary two-arm comparison.

**Self-rescue as a descriptive screen.** Among 23 measurements, 0 of the 12 strictly above a self-rescue rate of $0.182$ are significantly positive, and 4 lie exactly at that rate. This is an in-sample summary: the cut is defined as the largest rate among significantly positive measurements, so the absence of positive measurements above it follows by construction. It changes to $0.155$ under Bonferroni and $0.182$ under Benjamini–Hochberg, with 0 of 16 and 0 of 12 positive measurements above the respective cuts (Tables 12 and 13). Validation on new data is needed before using any cut predictively.

No measurement occupies the $18.5\%$ of the observed range immediately above the uncorrected cut. Family and sample size also differ across it: all 12 rows above it are $\tau^2$ with $T \leq 74$, and all 10 BFCL rows lie at or below it with $T \geq 199$. BFCL's only arm above the cut (`gpt56`, rate $0.210$) is its least accurate model and never occupies the second slot, which is assigned to the pair's better member. Estimating self-rescue also requires the retry control's own rollouts; it cannot replace their cost.

We recommend reporting the same-model retry control, the second-slot model's self-rescue rate, and a comparison with the pool's best fixed two-slot policy alongside each cascade oracle. Retry accounts for $26\%$ to $600\%$ of the naive gain here, so omitting it can materially change the interpretation. For offline evaluation, report logging granularity as well; our support measurements cover $\tau^2$ only. A one-call dispatcher avoids the attempt mismatch, and its selection-corrected headroom should be evaluated separately (Appendix A.14).

## 9 LIMITATIONS

Two benchmark families and 5 models establish that the routing effect can change sign, but they do not identify the causes of that variation. The substituted assertion grader may favour the strong model (Lai et al., 2026). Its exposure is uneven: $33\%$ pooled, all of `airline`, and none of `telecom` (Appendix A.1). On 45 grader-free `retail` training tasks, the effect is $+0.0444$ $[-0.0519, +0.1556]$, still covering zero. This check is unavailable on `airline`, whose null remains exposed to the grading concern.

All retries are blind restarts. Across 3 temperature configurations (0, 0.7, and one of each), effects range from $+0.0270$ to $+0.0360$, with 1 above the main estimate $+0.0315$. Temperature $0.7$ lowers one-slot reward from $0.6847$ to $0.6667$, so temperature does not order the controls. The strongest control gives $+0.0270$ $[-0.0270, +0.0878]$, still covering zero on `retail` train (Appendix A.3). Failure feedback could improve retry (Shinn et al., 2023; Chen et al., 2023b), although correction without an external signal is unreliable (Huang et al., 2024). Holding the cascade fixed, any improvement in the control's self-rescue rate reduces the routing effect. We did not evaluate feedback-conditioned policies. Continuing the cheap model's trajectory instead of restarting may also incur a handoff cost (Ganz et al., 2026); this deployment setting is outside our experiment.

The judge uses one implementation and the router one feature class (Appendices A.7 and A.17), so their failures do not establish that transcripts or task text are uninformative. The identity assumes binary reward, verified on every rollout; partial credit requires the general form in Appendix A.8. The step-level support analysis uses $K = 2$ for the reported corpus comparisons. Models are hosted endpoints whose weights and serving stacks we cannot inspect, and the observed nondeterminism may change with the serving path. We did not test benchmark contamination or compare routing with distilling the strong model into the cheap one (Gu et al., 2023), which could reduce the first-slot capability gap (Appendix A.22).

## 10 Related Work

**Routing between models.** Recent routers exploit execution feedback (Zhou et al., 2026b), live-traffic capability profiles (Li et al., 2026; Liu et al., 2026b), co-evolved prompts (Huang et al., 2026) and budget pacing (Taberner-Miller, 2026), or route context rather than models (Liu et al., 2025); bandit theory supplies the monotone-payoff regularity (Metelli et al., 2022). We ask what the oracle they target contains rather than proposing a better router, extending Zhou et al. (2026a). Our critique reaches *reactive cascades*, not predictive dispatchers committing to one model and one call; Appendix A.14 measures, rather than disputes, the oracle those aim at.

**Matched budgets and retry controls.** Prior work compares repeated attempts at matched budgets (Brown et al., 2024; Snell et al., 2024) and separates oracle headroom from deployable performance (Zhou et al., 2026a). We apply these principles to cascade evaluation on two agent benchmark families, where a second attempt is a whole episode, not a resampled answer; we add the same-model-retry control, which no cascade or router paper we audited reports at matched attempts (Appendix A.23); and we give the identity that says when the residual changes sign. Yang et al. (2026a) is closest, overlapping one of our families: it labels steps cheapest-sufficient and measures a router's deployed value, whereas we ask whether the offline weighting used to select one is usable at all (Section 7). Cost-aware cascades make the budget the treatment (Chen et al., 2023a; Jitkrittum et al., 2023); we hold it fixed, and our blind retry is the weakest member of a family whose stronger members condition on the first attempt's failure (Shinn et al., 2023; Chen et al., 2023b).

**Which model wins is domain-specific.** A narrow capability gap $p_s - p_c$ shrinks the first-slot cost, *helping* cheap-first against the intuition that routing pays most where models differ most; harness-repair work reports the same domain-dependence (Xu et al., 2026; Chen et al., 2026a); Appendix A.24 discusses further related work.

## Ethics Statement

The experiments involve no human subjects, personal data or crowd workers. Every task is a synthetic customer-service or tool-use scenario from a public benchmark; models interact with simulators or scripted environments. The benchmarks have permissive licences, listed in the supplementary archive's `NOTICE` alongside our handler and patches. We redistribute neither benchmark task data nor model weights. All 6,396 rollouts used five commercial inference endpoints under their providers' terms of service. The practical risk examined here is misattributing the benefit of another attempt to model routing, which can lead to paying for a cascade without evidence that model choice improves quality.

## Reproducibility Statement

Every measured quantity is emitted from stored result files by a script that refuses to build on an undefined macro, an uncited bibliography entry, or a hand-typed decimal in running prose or in inline mathematics; bare integers inside mathematics are not scanned, and the Use of AI section below names the exempt categories (Appendix A.25).

## References

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-Bench: Evaluating conversational agents in a dual-control environment. arXiv preprint arXiv:2506.07982; software at commit `672227c`, 2025.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023a.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, and Qing Wang. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026a.

Xiaoyi Chen, Siyuan Tang, Rui Zhu, Shijun Yan, Lei Jin, Zihao Wang, Liya Su, Zhikun Zhang, XiaoFeng Wang, and Haixu Tang. The Janus interface: How fine-tuning in large language models amplifies the privacy risks. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024. doi: 10.1145/3658644.3690325.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023b.

Zheng Chen, Hanqing Liu, Dulin Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic transactions for tool-using LLM agents. *arXiv preprint arXiv:2606.17573*, 2026b.

Yijiang River Dong, Tiancheng Hu, Zheng Hui, Caiqi Zhang, Ivan Vulic, Andreea Bobu, and Nigel Collier. Value of information: A framework for human-agent communication. *arXiv preprint arXiv:2601.06407*, 2026.

Roy Ganz, M. S. Nacson, Aditya Kalyanpur, and Ron Litman. The handoff tax: Continuing non-native trajectories in LLM agents. *arXiv preprint arXiv:2608.24358*, 2026.

Ashritha Gonuguntla. The replay gap: Static evaluation of model switching in LLM agents scores the wrong world. *arXiv preprint arXiv:2608.08239*, 2026.

Brandon Gower-Winter, Misja Groen, and Georg Krempl. The window dilemma: Why concept drift detection is ill-posed. *arXiv preprint arXiv:2602.06456*, 2026.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: On-policy distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Sina Hajimiri, Masih Aminbeidokhti, J. Dolz, I. Ayed, I. Laradji, Spandana Gella, and Nicolas Gontier. Are online skill and memory modules always worth their tokens? A budget-constrained study of web agents. *arXiv preprint arXiv:2606.15017*, 2026.

Yu He and Weikai Yang. SkillCommit: Evolving agent skills through behaviorally validated scope expansion. *arXiv preprint arXiv:2608.15165*, 2026.

Jiatan Huang, Zheyuan Zhang, Kaiwen Shi, Yanfang Ye, and Chuxu Zhang. EvolveR-outer: Co-evolving routing and prompt for multi-agent question answering. *arXiv preprint arXiv:2604.05149*, 2026.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *Proceedings of ICLR*, 2024. arXiv:2310.01798.

Yeonjun In, Md Mehrab Tanjim, Jayakumar Subramanian, Sungchul Kim, Uttaran Bhattacharya, Wonjoong Kim, S. Park, Somdeb Sarkhel, et al. Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation. *arXiv preprint arXiv:2603.25001*, 2026.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. *Proceedings of ICML*, 2016. arXiv:1511.03722.

Wittawat Jitkrittum, Neha Gupta, Aditya Krishna Menon, Harikrishna Narasimhan, Ankit Singh Rawat, and Sanjiv Kumar. When does confidence-based cascade deferral suffice? *arXiv preprint arXiv:2307.02764*, 2023.

Borui Kang, Jinrui Gu, Junhan Lv, Wen-Bin Li, Lei Wang, and Yang Gao. Harness continual learning: Continual adaptation beyond model parameters. *arXiv preprint arXiv:2608.19013*, 2026.

Peng Lai, Zhihao Ou, Yong Wang, Longyue Wang, Jian Yang, Yun Chen, and Guanhua Chen. BiasScope: Towards automated detection of bias in LLM-as-a-judge evaluation. *Proceedings of ICLR*, 2026. arXiv:2602.09383.

Rong Li, Ziyu Zhou, and Yihang Wu. FlyRoute: Self-evolving agent profiling via data flywheel for adaptive task routing. *arXiv preprint arXiv:2605.22057*, 2026.

Xurui Li, Kaisong Song, Rui Zhu, Pin-Yu Chen, and Haixu Tang. Adversarial attack-defense co-evolution for LLM safety alignment via tree-group dual-aware search and optimization. *arXiv preprint arXiv:2511.19218*, 2025a.

Xurui Li, Wanghaijiao, Kaisong Song, Rui Zhu, and Haixu Tang. Knowledge-aware co-reasoning for multidisciplinary collaboration. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13604–13620, 2025b.

Yitao Li. Who drifted: the system or the judge? Anytime-valid attribution in LLM evaluation pipelines. *arXiv preprint arXiv:2606.15474*, 2026.

Hao-Ran Ling, Yuecheng Li, Zeyu Song, Jing Yao, Shuwen Kang, Chi Lu, Wenjin Wu, and Peng Jiang. RecHarness: A bandit-routed agentic harness for self-evolving recommender systems. *arXiv preprint arXiv:2607.29241*, 2026.

Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjekye, Hao Tang, et al. RCR-Router: Efficient role-aware context routing for multi-agent LLM systems with structured memory. *arXiv preprint arXiv:2508.04903*, 2025.

Kai Liu, Guojun Xiong, Weinan Zhang, and Sheng Tang. Autoregressive diffusion world models for off-policy evaluation of LLM agents. *arXiv preprint arXiv:2606.05558*, 2026a.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *Proceedings of NeurIPS*, 2018. arXiv:1810.12429.

Xinchen Liu, Hang Zhou, Yingjie Zong, Yuchuan Tian, Liuyang Song, Shuo Zhang, Yulong Li, Wei He, et al. Agentic routing: The harness-native data flywheel. *arXiv preprint arXiv:2607.11399*, 2026b.

Xinyao Ma, Rui Zhu, Zihao Wang, Jingwei Xiong, Qingyu Chen, Haixu Tang, L. Jean Camp, and Lucila Ohno-Machado. Enhancing patient-centric communication: Leveraging LLMs to simulate patient perspectives. *arXiv preprint arXiv:2501.06964*, 2025.

Huanzhi Mao, Shishir G. Patil, and Sky Computing Lab. Berkeley Function Calling Leaderboard (BFCL). Software, `bfcl_eval` version 2026.3.23, 2026. `https://github.com/ShishirPatil/gorilla/tree/main/berkeley-function-call-leaderboard`.

A. Metelli, F. Trovò, M. Pirola, and Marcello Restelli. Stochastic rising bandits. *Proceedings of ICML*, 2022. arXiv:2212.03798.

Manvendra Modgil. The saturation trap and the subjectivity of intervention timing: Why affect-based triggers and LLM judges fail to time interventions on autonomous agents. *arXiv preprint arXiv:2606.04296*, 2026.

Johannes Moll, Jean-Philippe Corbeil, Jiazhen Pan, M. Hadamitzky, D. Rueckert, Lisa Adams, and K. Bressem. GRASP: Gated regression-aware skill proposer for self-improving LLM agents. *Proceedings of EMNLP*, 2026. arXiv:2605.29668.

Zhenhang Nie, Gui Zheng, Xudong Sun, Tailong Zhu, and Bin Zhang. AstronOS: A unified execution model and runtime for long-horizon agentic systems. *arXiv preprint arXiv:2608.16381*, 2026.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*, 2023.

Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 759–766, 2000.

Tom Saliencro, Rohan Desai, Priya Nair, Maya Lindqvist, and D. Whitmore. Uncertainty is not enough: Value-of-information routing for mixtures of LoRA experts. *arXiv preprint arXiv:2608.02528*, 2026.

Zayx Shawn. PACE: Anytime-valid acceptance tests for self-evolving agents. *arXiv preprint arXiv:2606.08106*, 2026.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Proceedings of NeurIPS*, 2023. arXiv:2303.11366.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Annette Taberner-Miller. ParetoBandit: Budget-paced adaptive routing for non-stationary LLM serving. *arXiv preprint arXiv:2604.00136*, 2026.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, A. Tomkins, Da-Cheng Juan, and Tu Vu. WikiSkill: Compiling agent experience into persistent knowledge for skill evolution. *arXiv preprint arXiv:2608.27454*, 2026.

Chen Xiong, Zihao Wang, Rui Zhu, Tsung-Yi Ho, Pin-Yu Chen, Jingwei Xiong, and Haixu Tang. Hey, that's my data! token-only dataset inference in large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 7105–7120, 2026.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic LLM agents. *arXiv preprint arXiv:2605.22166*, 2026.

Pei Yang, Wanyi Chen, Tong Yang, Pengbin Feng, Jiarong Xing, Wentao Guo, Yuhang Yao, Yuhang Han, et al. TwinRouterBench: Fast static and live dynamic evaluation for realistic agentic LLM routing. *arXiv preprint arXiv:2605.18859*, 2026a.

Xi-Dong Yang, Xingyi Zhang, Wen-Hao Li, Wen-Ya Liu, Junjie Sheng, Yun Hua, Wei Yin, Tao Fang, et al. PATH-Bench: Path-dependent evaluation of lifelong agents. *arXiv preprint arXiv:2608.01149*, 2026b.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. arXiv preprint arXiv:2406.12045, 2024.

Jianxiang Yu, Jiapeng Zhu, Bochen Lin, Qi Cui, Zichen Ding, and Xiang Li. Skill is not one-size-fits-all: Model-aware skill alignment for LLM agents. *arXiv preprint arXiv:2605.30723*, 2026.

Kai Zhang, Rui Zhu, Shutian Ma, Jingwei Xiong, Yejin Kim, Fabricio Murai, and Xiaozhong Liu. KEDRec-LM: A knowledge-distilled explainable drug recommendation large language model. *arXiv preprint arXiv:2502.20350*, 2025.

Congjie Zheng, Chuanyi Xue, Bin Liang, Jun Yang, and Changshui Zhang. SEAGym: An evaluation environment for self-evolving LLM agents. *arXiv preprint arXiv:2606.17546*, 2026.

Kaiji Zhou, V. Kalmykov, and Yue Feng. RouteRec: Strict evaluation of recommender-agent selection and aggregation. *arXiv preprint arXiv:2607.09908*, 2026a.

P. Zhou, Zhiwei Tang, Yixin Ma, Jiasheng Tang, Yizeng Han, Zhenglin Wan, Fanqing Meng, Wei Wang, et al. Agent-as-a-Router: Agentic model routing for coding tasks. *arXiv preprint arXiv:2606.22902*, 2026b.

Rui Zhu, Genevieve A. Mortensen, Yongming Fan, and Haixu Tang. Illuminating the unseen: A large-scale exploration of bias in ICU discharge summaries via language models. *IEEE EMBS International Conference on Biomedical and Health Informatics (BHI)*, 2025.

## USE OF AI

This paper was produced by an autonomous coding-and-research agent operating a research pipeline under human direction that set the topic, the acceptance criteria and the budget. The agent wrote all experiment code, ran all experiments, performed the analysis and drafted this text. The version-controlled record documents the following decisions. No acceptance threshold was edited. Of the five criteria, three were written and committed before their experiments ran and all three *failed*; the two that passed were inherited from a prior probe and appear already marked as passed in the first commit of the idea card, so we claim no pre-registration credit for them (Appendix C gives the full accounting). We report the failures rather than revising the criteria, and Appendix C also lists the negative results and the two discarded runs. The work was subjected to repeated adversarial review passes, each carried out by a separate instance of the same agent, given the paper, the analysis code and the raw rollouts, and instructed to refute each claim; where a pass found a claim unsupported, the claim was changed and the experiment extended, These reviews used raw transcripts that are not included in the archive. We include the per-rollout reward matrix (`results/rollouts.csv`) so readers can recompute the reported intervals without those transcripts. Every number in the tables and every measured quantity in the text is generated from stored result files by a script, and the generator refuses to emit if any macro the text references is undefined, if a bibliography entry is uncited, or if a numeral in the prose is hand-typed, or if a hand-typed decimal sits inside inline mathematics; bare integers inside mathematics are not scanned. Three categories of numeral are written literally and are not generated: experimental settings we chose (such as step caps, seeds, temperatures, confidence levels, hyperparameter grids and bootstrap counts), structural constants, and values produced by a guard probe rather than by a run, which exist to show that a check fires and therefore have no result file to come from. Every decimal literal in the first and third categories is allowlisted individually, with its reason, in the generator, and the generator prints the whole allowlist on each build; structural constants are not enumerated one by one.

The supplementary material carries the analysis code, the result file behind every table, a flat per-rollout reward matrix for all 6,396 rollouts of both benchmark families, and a file-level cryptographic digest of every raw rollout file. Those digests are per file, not per rollout: they establish that a copy of the rollout directory is ours, and the reward matrix is what lets an interval be recomputed.

We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including its text, claims and artifacts.

## A EXPERIMENTAL DETAIL

### A.1 BENCHMARK CONFIGURATION

| run tag | served endpoint | temp. |
|---|---|---|
| `haiku45` | `us.anthropic.claude-haiku-4-5-20251001-v1:0` | 0 |
| `sonnet45` | `us.anthropic.claude-sonnet-4-5-20250929-v1:0` | 0 |
| `opus45` | `us.anthropic.claude-opus-4-5-20251101-v1:0` | 0 |
| `gpt56` | `us.openai.gpt-5.6-terra` | provider default |
| `qwen3` | `qwen.qwen3-next-80b-a3b` | 0 |
| `glm5` | `zai.glm-5` | 0 |

Table 2: The served endpoint behind every run tag, and the decoding temperature each accepted. Generated by parsing the launcher, so the table cannot drift from what was called. One arm, `gpt56`, rejects the `temperature` field, which is why it is reported throughout as an unpinned arm throughout the analysis.

We use $\tau^2$-bench at commit `672227c`, with domains `retail`, `telecom` and `airline`. The agent implementation is `llm_agent`; the `user_simulator` uses the cheap model in every arm and domain. Fixing the simulator removes variation in simulator identity between arms but does not remove its influence on outcomes. On the cheap arm, the same model occupies both sides of the dialogue. The simulator uses the harness's default decoding configuration because the launcher

does not pass `--user-llm-args`. It is present in every $\tau^2$ arm and absent from BFCL, so its contribution to the cross-family difference cannot be identified separately. Section C describes an untested intervention.

Agents run at temperature 0, except `gpt56`, with three trials per (task, model) cell and run seed 300. The exception rejects the temperature field and uses the provider default: its `info.agent_info.llm_args` record is empty, whereas the other 4 arms record `temperature: 0.0`. Decoding can change self-rescue, so the exception matters to the interpretation. `gpt56` is both the most accurate model in the pool and the one with the highest self-rescue rate. We therefore report the best pool policy over all five arms and over the 4 pinned arms separately; the selected policy types differ (Section 5). The per-trial index does not reach the endpoint: the agent has no seed hook and the client drops unsupported parameters. Repeated trials thus measure endpoint nondeterminism, which may differ on another serving stack.

Episodes are capped at 50 steps on `retail` and `airline` and at 100 on `telecom`; Section A.5 explains why `telecom` differs and what the 50-step pass would have implied. Across the $\tau^2$ runs (2,196 of the 6,396 rollouts in the corpus; BFCL has neither a user simulator nor a step cap, so this inventory does not reach it), 2,167 end with the user simulator stopping normally, 28 reach the step cap and 1 ends in an infrastructure error. That rollout is scored as a failure rather than discarded, but a second rule then removes its whole task, and the two together are how `telecom` reaches the 73 tasks Table 1 reports rather than the 74 it was launched with. The task-selection rule is: for each task take the minimum trial count across arms, take the mode of those minima as $n$, and drop any task with fewer than $n$ usable trials. So a rollout lost to an infrastructure error costs its task all three trials, in every arm. Scoring the failed rollout without applying task selection would retain 74 tasks and produce a different estimate.

BFCL uses a different missing-data rule. A cell whose result begins with an inference error is set to missing and the task is dropped listwise across all arms, taking the pool from 200 tasks to 199. The two benchmarks therefore use different task-selection rules. The erroring cell belongs to an arm that appears only in the pool study, which is why the two-arm BFCL rows of Table 1 keep all 200 tasks while the 10 pool pairs use 199: the listwise drop is over the arms a comparison actually includes. Table 3 counts how every rollout ended, run by run.

We patched four things in the vendored benchmark, all recorded in version control. (i) The episode cap: an earlier pass used a cap tight enough that most rollouts ended at it, so the measured arm difference reflected finishing speed rather than capability. That pass is not archived, so Appendix C states the fact without quoting its numbers. (ii) `DummyUser.__init__` rejects a keyword argument the runner passes in solo mode. (iii) Three default grader and interface models pointed at providers we cannot call; we redirected them to a single fixed model, identical across arms. Assertion-graded exposure differs substantially across domains: 29 of 74 `retail` training tasks, 30 of 30 on `airline`, and 0 of 74 on `telecom`, or 33% pooled; the held-out `retail` test split carries 11 of 40. On `retail` 39% of reward passes through a language-model grader, and the grader we substituted is the same model as the strong arm.

Self-preference by this grader could favour the retry baseline. Retry uses strong-model rollouts in both slots, whereas the cascade uses one. Under the stated condition $p_s - p_c < 1 - p_s$, which holds on every graded dataset (Table 11), extra credit to strong-model rollouts increases $\mathrm{V}(s \to s)$ faster than $\mathrm{V}(c \to s)$. Such a bias would lower the measured routing effect and could support the observed nulls on `retail` and `airline`.

We assess this risk on tasks without assertion grading. On the 45 grader-free `retail` training tasks, the effect is $+0.0444$ $[-0.0519, +0.1556]$, compared with $+0.0315$ $[-0.0360, +0.1081]$ on all 74 tasks. On the held-out split it is $-0.0057$ $[-0.0460, +0.0345]$, compared with $-0.0042$. The point estimates move in the predicted direction, but the intervals still cover zero. `telecom` has no assertion-graded tasks, so this grader cannot explain its negative effect. All `airline` tasks use the grader, leaving no grader-free subset for that domain. Reporting exposure by domain is necessary because pooled rates can hide unequal grading coverage, as also observed in clinical-document evaluation (Zhu et al., 2025).

(iv) The assertion grader parsed raw model output with `json.loads`, which failed when responses included code fences. We made parsing tolerant of those fences without changing the grading criteria.

| run | rollouts | user stop | step cap | infra. error |
|---|---|---|---|---|
| retail train | 444 | 441 | 3 | 0 |
| `sonnet45` | 222 | 219 | 3 | 0 |
| `haiku45` | 222 | 222 | 0 | 0 |
| telecom train | 444 | 436 | 7 | 1 |
| `sonnet45` | 222 | 215 | 6 | 1 |
| `haiku45` | 222 | 221 | 1 | 0 |
| airline train | 180 | 178 | 2 | 0 |
| `sonnet45` | 90 | 88 | 2 | 0 |
| `haiku45` | 90 | 90 | 0 | 0 |
| retail test | 240 | 240 | 0 | 0 |
| model pool | 666 | 653 | 13 | 0 |
| `qwen3` | 222 | 211 | 11 | 0 |
| `gpt56` | 222 | 220 | 2 | 0 |
| `glm5` | 222 | 222 | 0 | 0 |
| retry $T$=0.7 | 222 | 219 | 3 | 0 |
| $\tau^2$ subtotal | 2196 | 2167 | 28 | 1 |
| BFCL `multi_turn` $T$=0 | 3000 | n/a | n/a | 1 |
| BFCL `multi_turn` unpinned | 1200 | n/a | n/a | 0 |
| corpus | 6396 | 2167 | 28 | 2 |

Table 3: How every one of the corpus's 6,396 rollouts ended, by run. Only $\tau^2$ has a user simulator and a step cap, so those two columns read n/a for BFCL rather than 0: a zero would assert that no rollout hit a cap that does not exist. BFCL's one infrastructure error costs its task listwise, which is why the pool is 199 of 200 tasks; $\tau^2$'s is initially scored as a failure, but its task is then removed from reward analysis under the trial-count rule.

### A.2 BFCL CONFIGURATION AND WHAT WE CHANGED

BFCL v4 (Mao et al., 2026; Patil et al., 2023), category `multi_turn_base`, first 200 entries in the released order, 3 trials per (task, model) cell. Correctness is the harness's own state check: after the model's tool calls execute, the resulting Python object state is compared against the state produced by the ground-truth call sequence, so a run can fail with the message that a file-system instance does not match ground truth even when every call individually parsed. There is no language-model user simulator; multi-turn structure comes from a fixed script of user messages. This removes user-simulator stochasticity from BFCL, although other differences between the benchmarks prevent a causal attribution to the simulator alone.

We report two BFCL passes, labelled pinned (primary) and unpinned (secondary).

**The secondary pass (unpinned $T$), and why it exists.** The first pass reused the harness's Anthropic handler with the Bedrock-backed client of the same SDK. We changed three things in a vendored copy, none of them touching grading. (i) The client is swapped for the Bedrock-backed one, because that is the endpoint we have; request and response shapes are identical. (ii) The `temperature` field is dropped. The restriction comes from the SDK, rather than an endpoint rejection: its `messages.create` for the Bedrock backend at the pinned version does not accept a `temperature` parameter at all for this model generation, so the call raises a `TypeError` before reaching the endpoint. *On this path no temperature can be set*, and both arms sampled at the provider default. (iii) Two model identifiers are registered as function-calling models with the harness's underscore-to-dot name conversion. This pass gives the row labelled *unpinned $T$* in Table 1: routing $+0.0467$ $[+0.0150, +0.0808]$.

**The primary pass (pinned $T$), and the handler it needed.** Because "more stochastic" is precisely the variable our mechanism claim concerns, we could not leave the decoding tem-

perature unpinned. We therefore wrote a fourth vendored change: a new handler that calls Bedrock's `converse` API directly rather than going through the Anthropic SDK, and sets `inferenceConfig.temperature` $= 0$. This also changes several aspects of the call path: the API surface moves from `messages.create` to `bedrock-runtime.converse`; the harness's model style moves from the Anthropic serialisation to the endpoint's native one, which changes how tool schemas and results are encoded; prompt caching is not requested on this path; reasoning blocks arrive under a different key, which we parse tolerantly because not every model in the pool emits the same sub-fields; and the output-token limit is left at the endpoint's default rather than set, which we checked rather than assumed away (across the pass's generation calls the largest single output is far under any plausible default, so no response was truncated by it). Section 5 treats this bundle as a candidate cause of the BFCL difference and bounds it with the secondary pass rather than dismissing it. The primary two-arm row is therefore pinned at $0$ throughout.

Both two-arm models accept the field. In the $5$-model pool of Section 5, $4$ of $5$ arms accept it and one does not, so that arm ran at the provider default; the pinning state of every arm is read from the handler registry and the generation logs and recorded in the stored results rather than asserted here.

**Robustness of the primary pass.** Reducing each (task, arm) cell to the majority of its three trials (which removes trial-to-trial variance entirely) gives a routing effect of $+0.1150$, *larger* than the headline $+0.0867$. Resampling tasks *and* trials hierarchically instead of tasks alone gives $[+0.0425, +0.1375]$ under the common shock, which still excludes zero. Per-task non-unanimity is $0.000$ for the cheap arm and $0.090$ for the strong one; the strong arm is *more* reproducible here than on $\tau^2$ `retail` ($0.311$), which is the observation that motivated the user-simulator hypothesis, since language-model-simulated users are by now a standard evaluation instrument (Ma et al., 2025), and also the reason we do not claim it as identified (Section C).

**What bounds the call path, since the rerun changed it too.** The primary pass changed the decoding temperature and the API surface at once, so on its own it cannot separate the two. The secondary pass is the control for that, and it is a useful one because it is *not* ours: it runs the harness's own handler with only the client object swapped, and it still gives a significantly positive routing effect ($+0.0467$ $[+0.0150, +0.0808]$). On that pass the strong arm's self-rescue rate has already fallen from $\tau^2$ `retail`'s $0.3286$ to $0.1620$, which is $72.5\%$ of the whole drop to the pinned pass's $0.0989$. The remaining $27.5\%$ of the observed rate change occurs between the two BFCL passes, where temperature and handler change together. The positive sign is present under both handlers; this arithmetic does not identify either change's causal contribution. A cross-path capability check points the same way: the cheap arm scores $0.6417$ on the secondary pass and $0.6750$ on the primary one, and the strong arm $0.7017$ against $0.6967$, so the handler did not move either arm's accuracy far. What we do not have is an external leaderboard anchor for either pass.

We ran the harness's own scorer unmodified. Evaluating a subset requires the harness's `-partial-eval` flag; without it the scorer refuses, to distinguish subset evaluation from scoring the full split.

**Cost, and which pass each caveat belongs to.** The passes use different token-logging regimes. On the *secondary* pass the harness records the API's reported `input_tokens` with prompt caching enabled for the multi-turn categories, so cache-creation and cache-read tokens are excluded from the log entirely. The scale of that omission is visible in the ratio: logged input tokens are $0.080$ times output tokens ($0.079$ on the cheap arm, $0.080$ on the strong one), consistent with omitted cached prompt tokens. Both arms omit cached tokens, so the secondary pass underestimates absolute cost. Its logged cost ratio should also be interpreted cautiously because omitted token counts may differ by arm. The *primary* pass is not in that regime at all: its handler does not request prompt caching, so the whole prompt is logged on every turn and the ratio is $72.08$ ($70.90$ and $73.26$ per arm), nearly three orders of magnitude higher. The one BFCL cost figure the main text quotes, $1.95\times$, comes from that primary pass, so the lower-bound caveat does not reach it. The stored cost note is computed separately for each pass and arm from its measured token ratios. The generator checks that the two passes remain separated by an order of magnitude on each side of one.

**The price basis.** Every dollar figure in this paper uses one table: the litellm bundled model_prices_and_context_window map at version 1.82.6, keyed on the served endpoint identifier

| retry control | control reward | routing effect | 95% CI |
|---|---|---|---|
| both slots $T$=0 (main text) | 0.7883 | +0.0315 | [−0.0360, +0.1081] |
| both slots $T$=0.7 | 0.7838 | +0.0360 | [−0.0315, +0.1081] |
| $T$=0 → $T$=0.7 | 0.7928 | +0.0270 | [−0.0270, +0.0878] |

Table 4: The routing effect against each of the three retry controls, on the `retail` training split. *control reward* is the retry baseline's own two-slot reward; the main text's decomposition uses the first row.

of Table 2, which prices the cheap arm at \$1 per million input tokens and \$5 per million output, and the strong arm at \$3 and \$15. The two families reach it differently. BFCL logs token counts and our analysis multiplies, and only the *ratio* of those two rows enters any reported number. $\tau^2$ multiplies inside the benchmark and stores only the product: its run records carry costs in dollars and no usage field at all, so a reader can check the arithmetic of our cost ratios but cannot re-price them, and a different version of that pricing library would silently re-price the whole $\tau^2$ cost column. That is why the library is pinned alongside the harnesses in `code/requirements.txt` rather than left to resolve.

### A.3 A STRONGER RETRY CONTROL: TEMPERATURE 0.7

The retry control in the main text retries the strong model at temperature 0, which introduces no deliberate sampling diversity: at temperature 0 a second attempt differs only through endpoint nondeterminism. To assess the effect of sampling diversity, we ran the strong model a third time on the `retail` training split at temperature 0.7 and re-derived the decomposition against three retry baselines: both slots at temperature 0, both at 0.7, and 0 then 0.7. The comparison uses the same ordered-pair accounting throughout.

Sampling at 0.7 lowers one-slot reward from 0.6847 to 0.6667 (Table 4). The three routing effects range from +0.0270 to +0.0360, compared with +0.0315 under the temperature-0 control. Of the two variants, 1 increases the effect. The full span is 0.0090, roughly balanced around the main estimate, so these results show no monotonic response to temperature. The strongest control gives +0.0270 [−0.0270, +0.0878]; the effect is significantly positive against 0 controls. This experiment covers `retail` train only. BFCL's positive result has not been tested against a retry control with temperature above 0; its pinned/unpinned comparison changes both arms as well as the API path.

For a fixed cascade and first-attempt distribution, a more successful retry lowers the routing effect. In the identity, increasing self-rescue by $\delta$ reduces the effect by $(1 - p_s)\delta$. Changing both retry slots can also change $p_s$, so temperature alone need not order the controls. This explains why the strongest control lowers the effect while using 0.7 in both slots raises it.

**Blind retries and failure feedback.** All tested retries restart the task without information about the first attempt. Feedback could improve the control: possible variants use the first transcript (Shinn et al., 2023), an environment error trace (Chen et al., 2023b), or a self-critique between attempts. The first two have empirical support; self-correction without an external signal is less reliable (Huang et al., 2024). None is guaranteed to improve self-rescue. To the extent that a feedback-conditioned retry improves it while holding the cascade fixed, our measured routing effect is an upper bound on the resulting comparison. A matched feedback experiment should also let the cascade's second model see the first attempt's feedback. That requires four policy conditions, which we did not run.

### A.4 TWO-SLOT ACCOUNTING AND THE DECOMPOSITION

For trial indices $\mathcal{I}$ with $|\mathcal{I}| = 3$ we evaluate every two-slot policy over all 6 ordered pairs $(i, j)$, $i \neq j$. A policy $k_1 \to k_2$ scores $r_{t,k_1,i}$ if $r_{t,k_1,i} \geq 1$ and $r_{t,k_2,j}$ otherwise, and pays $c_{t,k_1,i}$ always plus $c_{t,k_2,j}$ only when it escalates. One-slot policies use a single trial, averaged over the three. Confidence intervals are 10,000-sample bootstraps resampling *tasks*, with the policy and its comparator recomputed on the same task draw, so the intervals are paired.

This accounting is what the retry control rests on: both the cascade and the same-model retry use the same two-slot budget, and neither is ever scored against the attempt used to trigger it.

### A.5 TRUNCATION AND TELECOM

A first telecom pass used the same 50-step cap as the other domains and truncated 243 of 444 rollouts, against 3 of 444 on retail. That pass is superseded. Raising the cap to 100 steps and rerunning all 74 tasks × 2 arms × 3 trials brought truncation down to 7 of 444 rollouts, and the 100-step data is what Table 1 and every telecom number in the main text report.

Restricting the earlier 50-step pass to the 14 of 74 tasks whose six rollouts all terminated normally gives an effect of exactly 0. This restriction selects tasks using an outcome-related criterion and does not establish that truncation caused the negative effect. With the 100-step cap and all 73 usable tasks, the effect is $-0.0913$ $[-0.1279, -0.0548]$: larger in magnitude and still significantly negative. The retained truncated pass is therefore a sensitivity analysis, not the basis of the main result.

The identity explains this negative effect through the capability gap. The cheap arm succeeds on $0.2192$ of tasks and the strong arm on $0.6073$, giving a first-slot cost of $-0.3881$ that the rescue term does not offset (Section 4).

### A.6 WHICH TERM OF THE IDENTITY ACTUALLY VARIES

**Estimators in Figure 3b.** The figure uses different escalation signals for the oracle and deployable policies. All five series *score* on the same two evaluation trials, and the third trial is never scored. What differs is where the gate reads. The oracle cascade takes its failure signal from that third trial, so it is a genuine split-half estimator and $0.6757$ is always-strong under it. The 112 observable proxy rules and the 3 judge cascades take their signal from the same trial they score, because that is what a deployed system has: it decides from the attempt in front of it. This matches deployment, but changes the estimator. Under the split-half gate, the best proxy on this dataset rises to $0.6824$, with 4 of the 112 configurations clearing always-strong instead of none; on the other three datasets it falls instead, so 1 of 4 datasets improves under this change. Either way 0 is a statement about the as-deployed gate and not about proxy rules in general, and the figure draws the as-deployed one because that is the claim Section 6 makes. The $0.6847$ of Section 4 is a third estimator again, averaging all three trials; it differs from $0.6757$ by $-0.0090$ here and by up to $0.0208$ across our 4 datasets, changing sign between them. The cost axis is relative to always-escalate, the cheap call on every task plus the strong call.

**The retry reference for Figure 3b.** Every series in Figure 3b is gated against one-slot always-strong, which is the unequal comparison Section 3 rejects, and the obvious repair is to add the two-slot retry control. It cannot be added under the split-half gate, for a reason worth stating: that gate does not depend on which evaluation trial fills the first slot, so choosing between the two evaluation trials is a relabelling, and averaging over both makes a retry policy identically equal to always-strong in reward while costing more. The resulting retry has the same reward at $1.050$ times always-escalate's cost, a gain of exactly zero. A retry policy carries information only when the gate reads the first slot's own rollout, which is the main text's convention and is legitimate there because the attempt that is *scored* is still a different rollout from the one that triggered it. Computed that way on these tasks the retry control reaches $0.8108$ and the matched cascade $0.8041$, a difference of $-0.0067$ that is this section's own estimate of the routing effect on two of three trials and sits inside Table 1's interval for it. Against those two points, 0 of the judge tiers reach the retry control and 0 reach the cascade. We do not draw them on the figure, because the oracle uses a split-half gate, while these retry and cascade references use the main text's ordered-pair convention.

**The governing axis in full.** Table 16 puts every measurement on one axis and Figure 3a plots it: 23 measurements of the routing effect against the second slot's self-rescue rate, with 2 nominal points dropped as repeat scorings of a comparison already present. We distinguish unweighted rescue rates from the weighted terms in the identity: the identity weights the two rescue terms by $(1 - p_c)$ and $(1 - p_s)$, and in those units the self term is not the most variable of the three. The self-rescue column spans $0.526$ while the cross-rescue column stays inside $0.247$: the self-rescue rate therefore spans more than twice the range of the cross-rescue rate. And because the rate is a property of

| the published axis, by aggregation level | | | | $\rho$ against each candidate term | | | |
|---|---|---|---|---|---|---|---|
| level | $n$ | $\rho$ | 1-sided $p$ | clusters | self | gap | cross |
| cells | 11 | $-0.84$ | 0.001 | all cells | $-0.84$ | $-0.69$ | $-0.66$ |
| datasets | 5 | $-0.40$ | 0.258 | $\tau^2$ only | $-0.43$ | $-0.61$ | $-0.50$ |
| $\tau^2$ cells | 7 | $-0.43$ | 0.177 | BFCL only | $-0.80$ | $-1.00$ | $-0.80$ |
| BFCL cells | 4 | $-0.80$ | 0.167 | | | | |

Table 5: Left: the published axis at every aggregation level the corpus admits, with the one-sided permutation $p$ at each. Right: the same clusters, correlating the routing effect against each of the identity's three candidate terms, pooled and within each family. *cells* are (dataset, second-slot model) pairs.

the second-slot *model*, the 23 rows cover only 11 distinct (dataset, second-slot model) cells, so we correlate at cluster level rather than counting pairs that share a second slot twice: $-0.84$, permutation $p = 0.001$. Restricting to the 10 clusters with a capability gap below $0.15$, the restriction the identity asks for, raises it to $-0.96$; we report that as a sensitivity and not as the headline, because on this corpus it removes exactly one cluster and that cluster is `telecom`, our only significantly negative measurement. We fit no curve: 11 values identify a direction, not a shape.

**What the permutation test does and does not test.** The cluster-level permutation test asks whether the observed ordering could arise from an arbitrary pairing of the same numbers. It does not test independence, and it could not: the identity makes the two quantities dependent by construction, with the second slot's self-rescue rate entering the routing effect with coefficient $-(1 - p_s)$. Two calculations put a number on how much of the ordering that coefficient is responsible for. Remove the term the rate enters through, correlating the rate against routing $+ (1 - p_s)\mathbb{E}[s \mid s \text{ fails}]$, and the cluster correlation falls from $-0.84$ to $-0.33$ with a one-sided permutation $p$ of $0.16$. Or keep the identity intact and permute only the part the rate does not enter, which leaves each cluster's mechanical term attached to its own rate: the null is then centred on $-0.64$ and largely contained in $[-0.88, -0.34]$, and the observed $-0.84$ sits at $p = 0.11$. Thus $p = 0.001$ describes ordering under a permutation null; it does not establish a causal mechanism beyond the identity. The clusters also share tasks (all ten BFCL pool pairs run on the same 199 tasks and all ten $\tau^2$ pool pairs on the same 74), so the effective number of independent experiments behind the axis is smaller than 11, and we do not claim otherwise.

**The same correlation at every aggregation level, and for every competing term.** Table 5 gives both. The left block is the published axis at four aggregation levels: the 11 cells, the 5 datasets, and each family's cells alone. Only the cell level clears the conventional level, which is the sense in which the axis is established *between* the two benchmark implementations rather than inside either. The right block correlates the routing effect against the identity's other two terms on exactly the same clusters. Pooled, self-rescue leads at $-0.84$ against the capability gap's $-0.69$ and the cross-rescue rate's $-0.66$; *inside* $\tau^2$ the gap leads instead, $-0.61$ against $-0.43$, and at row level too ($-0.66$ against $-0.49$). A paired cluster bootstrap of the difference in $|\rho|$ gives $+0.14$ $[-0.48, +0.80]$ with the gap ordering better in $0.31$ of draws, so at this cluster count the two are not distinguishable. What picks self-rescue out is therefore not fit. The rate requires one extra rollout per task and no second model. The observed boundary in Section 8 is descriptive and requires validation on independent data.

**The within-$\tau^2$ correlation, and every leave-one-out.** Within $\tau^2$'s 7 clusters alone the correlation is $-0.43$, and it is unstable under dropping a single cluster, as shown by the full leave-one-cluster-out analysis. Dropping `telecom` raises it to $-0.94$; dropping the `haiku45` cluster instead takes it to $-0.09$, near zero. Table 6 gives all 7. That cluster is `retail` train's only significantly positive measurement, it is the only $\tau^2$ point below the cut of Section 8, and its second-slot model is also the fixed user simulator in every $\tau^2$ arm, Removing this cluster substantially weakens the within-$\tau^2$ association.

**Task versus trial resampling, and the multiplicity family, in full.** First, our intervals resample *tasks* and hold the three logged rollouts per cell fixed. Resampling trials too widens them $1.48\times$ on

| cluster dropped | rows | clusters left | $\rho$ |
|---|---|---|---|
| retail train / `haiku45` | 1 | 6 | $-0.09$ |
| airline train / `sonnet45` | 1 | 6 | $-0.37$ |
| retail train / `glm5` | 3 | 6 | $-0.37$ |
| retail train / `sonnet45` | 2 | 6 | $-0.37$ |
| retail test / `sonnet45` | 1 | 6 | $-0.43$ |
| retail train / `gpt56` | 4 | 6 | $-0.43$ |
| telecom train / `sonnet45` | 1 | 6 | $-0.94$ |

Table 6: Within-$\tau^2$ leave-one-cluster-out for the governing axis, every cluster, ordered from the least to the most favourable. *rows* is how many of the 13 $\tau^2$ measurements the dropped cluster holds.

average and changes no verdict in 4: `telecom`'s negative effect stays at $[-0.1507, -0.0274]$ and BFCL's positive effect at $[+0.0425, +0.1375]$ under the common shock, both still excluding zero. It does change the BFCL pool pairs `gpt56`→`glm5` and `qwen3`→`glm5`, so the count of significantly positive pool pairs reads 7 of 10 rather than 9 once the trials are freed. Second, the correction runs over exactly the 23 measurements the governing axis plots, one deduplication rule applied once, so the count we report and the count we correct over cannot diverge. 8 survive Bonferroni, including `telecom` and BFCL's two-arm result, both of whose adjusted $p$ sit at the reporting floor of 0.0023 that 20,000 resamples can resolve. Appendix A.13 has both tables.

Section 5 compares the self-rescue and cross-rescue *rates*, which are unweighted. The identity weights them: the rescue term is $(1 - p_c)\,\mathbb{E}[s \mid c \text{ fails}]$ minus $(1 - p_s)\,\mathbb{E}[s \mid s \text{ fails}]$, so a rate with a small failure mass in front of it moves the routing effect less than its spread suggests. In those units, across all 23 measurements, the standard deviations are 0.0776 for the first-slot cost, 0.0605 for the cross term and 0.0366 for the self term. The self term is the *least* variable of the three.

We state this because it bounds the interpretation of the association. The axis is a claim about *correlation*: the second slot's self-rescue rate is the quantity that orders the routing effect across benchmarks, and it does so at $-0.84$ over 11 clusters. It is not a claim that the self term carries the most variance, and a variance-decomposition reading of Section 5 would be wrong. The capability gap provides a comparably strong ordering. Self-rescue is useful because it can be estimated from one extra rollout per task without running another model.

### A.7 What the judge reads, and what it never sees

Section 6's judge cascade sends a rendered transcript to a language model and asks whether the agent finished the task. Four properties affect how these judge results should be interpreted.

First, the judge receives Chinese instructions and speaker labels with English dialogue. Its free-text reason is limited to twenty Chinese characters. The exact `PROMPT` template and renderer are included in `code/judge_cascade.py`; the transcript replaces the single `{convo}` field. In English, the instructions ask the judge to use only the dialogue and not infer database state. Promising an action without the corresponding tool call, completing only part of a request, or stalling without completion counts as failure. Correctly refusing a forbidden request while completing the rest counts as success. The response must be a JSON object with a boolean verdict and a reason of at most twenty characters. We did not measure how the cross-lingual prompt or reason limit affects accuracy.

Second, the transcript is truncated three times, not once: each tool call's arguments to 300 characters, each message body to 800, and the assembled transcript to 14,000. A judge that misses a late tool call because an earlier message consumed the budget will score a completed task as incomplete, and that error is not symmetric across arms, because the cheap arm produces longer transcripts.

Third, every judge tier is a model from the strong arm's own family: all 3 of `haiku45`, `sonnet45` and `opus45` share a vendor and a generation (Table 2), and on the `sonnet45` tier the judge *is* the strong arm.

Fourth, the 148 judgements are not 148 distinct transcripts: 56 of them are on a transcript byte-identical to another judgement's, because a deterministic arm that reproduces a rollout hands the

judge the same text twice. The intervals in Figure 4a are clustered on the task, which absorbs that repetition into the cluster, but the effective number of *distinct* decisions behind each tier's accuracy is smaller than its denominator suggests, which is the other reason we do not order the tiers.

These design choices limit generalisation from this judge to other transcript-based detectors.

**Limits of the judge experiment.** The results have at least two possible explanations: the transcript may omit information needed to judge the final database state, or our judge may use the available information poorly. The cross-lingual prompt, short reason field and truncation could each reduce accuracy; the cheap arm's longer transcripts make truncation asymmetric. The tier comparison cannot distinguish these explanations. Our result is that this judge fails to recover the verifier-gated gain, not that transcripts are inherently uninformative. An ablation with English instructions, unrestricted reasons and a transcript budget prioritising later turns would help separate these explanations; we did not run it.

### A.8 Derivation of the routing identity

Section 4 states the identity without derivation. Write $\nVdash_{t,k,i}$ for the event $r_{t,k,i} \geq 1$ and let $\mathcal{P}$ be the set of ordered trial pairs $(i, j)$, $i \neq j$. Under the two-slot accounting, for any pair of arms $k_1, k_2$,

$$\mathrm{V}(k_1 \to k_2) = \frac{1}{|\mathcal{P}|\,T} \sum_{(i,j)\in\mathcal{P}} \sum_{t} \Big[ \nVdash_{t,k_1,i} + (1 - \nVdash_{t,k_1,i})\, r_{t,k_2,j} \Big].$$

When reward is binary the first term averages to the one-slot success rate $p_{k_1}$ exactly, and the second is $(1 - p_{k_1})$ times the average of $r_{t,k_2,j}$ over precisely those $(t, i, j)$ with $\nVdash_{t,k_1,i} = 0$, which is the conditional expectation $\mathbb{E}[k_2 \mid k_1 \text{ fails}]$ under the empirical measure. Hence $\mathrm{V}(k_1 \to k_2) = p_{k_1} + (1 - p_{k_1})\,\mathbb{E}[k_2 \mid k_1 \text{ fails}]$, and subtracting the instance $k_1 = k_2 = s$ from the instance $k_1 = c$, $k_2 = s$ gives the identity. Both conditional expectations are computed on the same $\mathcal{P}$, so no reweighting is needed and the residual is numerical only; the largest we observe across the 4 datasets is $1e-16$.

The identity has three scope conditions. First, it needs binary reward: with partial credit the first term is $\mathbb{E}[r_{k_1} \nVdash]$ rather than $p_{k_1}$ and the decomposition acquires a third term. Our reward is binary on every rollout in every dataset (we check this and abort otherwise), but $\tau^2$-bench can award partial credit in principle, and a run configured differently would need the general form. Second, it is an identity about *our* escalation rule, the benchmark verifier; a noisy trigger changes both conditional expectations and the identity must be re-derived. Third, it is descriptive, not causal: $\mathbb{E}[s \mid c \text{ fails}] > \mathbb{E}[s \mid s \text{ fails}]$ says the strong model recovers the cheap model's failures more often, but the two conditioning events select different task populations, and part of the gap is that $c$ fails on easier tasks than $s$ does. That is precisely why we do not read the gap as a causal complementarity effect and do not build a router on it.

### A.9 All policies, all datasets

Table 7 gives every policy on every dataset at full precision, including the two that the main text does not use: `retry cheap`, which bounds what a budget spent entirely on the cheap model buys, and `cascade` $s \to c$, the reversed order. The reversed cascade matches $c \to s$ in reward on *every* dataset, which is an identity of the estimator rather than a fact about routing (Appendix A.26); it differs only in cost. Which order is cheaper follows from the price and accuracy ratios, not from escalation rates: see Appendix A.26 and Table 31.

### A.10 The four-scalar sign predictor in detail

Table 8 gives the rule's sign accuracy against the size of the profiling sample. On `retail` it rises only from $0.495$ to $0.590$ as the profiling set grows to half the tasks; that clears `retail`'s own always-retry baseline of $0.237$ at every sample size but stays under the $0.623$ those constants score averaged over the four datasets, and on `airline` and the held-out split the curve does not rise at all. The lack of improvement is consistent with unstable target signs, but it does not establish a limit on other estimators. On `telecom` the rule is already correct at $n = 10$.

| policy | reward | cost | escalation rate | reward per $ |
|---|---|---|---|---|
| *retail train* ($T = 74$, 3 trials, 6 ordered pairs) | | | | |
| always cheap | 0.6126 | $8.20 | 0.000 | 0.0747 |
| always strong | 0.6847 | $26.74 | 0.000 | 0.0256 |
| retry cheap | 0.6306 | $11.74 | 0.387 | 0.0537 |
| retry strong | 0.7883 | $36.14 | 0.315 | 0.0218 |
| cascade $c \to s$ | 0.8198 | $20.26 | 0.387 | 0.0405 |
| cascade $s \to c$ | 0.8198 | $29.47 | 0.315 | 0.0278 |
| *telecom train* ($T = 73$, 3 trials, 6 ordered pairs) | | | | |
| always cheap | 0.2192 | $25.56 | 0.000 | 0.0086 |
| always strong | 0.6073 | $60.48 | 0.000 | 0.0100 |
| retry cheap | 0.2374 | $47.48 | 0.781 | 0.0050 |
| retry strong | 0.7169 | $89.72 | 0.393 | 0.0080 |
| cascade $c \to s$ | 0.6256 | $75.83 | 0.781 | 0.0083 |
| cascade $s \to c$ | 0.6256 | $72.11 | 0.393 | 0.0087 |
| *airline train* ($T = 30$, 3 trials, 6 ordered pairs) | | | | |
| always cheap | 0.5667 | $2.90 | 0.000 | 0.1954 |
| always strong | 0.6444 | $11.42 | 0.000 | 0.0564 |
| retry cheap | 0.5667 | $4.57 | 0.433 | 0.1240 |
| retry strong | 0.7667 | $16.15 | 0.356 | 0.0475 |
| cascade $c \to s$ | 0.7667 | $9.28 | 0.433 | 0.0826 |
| cascade $s \to c$ | 0.7667 | $12.66 | 0.356 | 0.0606 |
| *retail test* ($T = 40$, 3 trials, 6 ordered pairs) | | | | |
| always cheap | 0.7917 | $4.24 | 0.000 | 0.1867 |
| always strong | 0.8417 | $13.13 | 0.000 | 0.0641 |
| retry cheap | 0.8000 | $5.10 | 0.208 | 0.1569 |
| retry strong | 0.9250 | $15.36 | 0.158 | 0.0602 |
| cascade $c \to s$ | 0.9208 | $7.37 | 0.208 | 0.1249 |
| cascade $s \to c$ | 0.9208 | $13.88 | 0.158 | 0.0663 |

Table 7: Every policy, every dataset. *escalation rate* is the fraction of first-slot attempts that trigger the second slot; it is a property of the first-slot arm alone. *reward per $* is included to make the cost comparison legible, not because it is the objective.

| dataset | $n = 10$ | $n = 20$ | $n = 30$ | $n = 40$ |
|---|---|---|---|---|
| retail train | 0.495 | 0.537 | 0.532 | 0.590 |
| telecom train | 0.990 | 0.998 | 1.000 | 1.000 |
| airline train | 0.102 | 0.070 | n/a | n/a |
| retail test | 0.385 | 0.255 | 0.343 | n/a |

Table 8: Rule sign accuracy against profiling-sample size $n$, with the remaining tasks used for evaluation. A cell reads n/a when the profiling size is at or above the dataset's task count.

Table 9 gives the transfer test. The prediction depends only on the four scalars estimated on the fitting dataset, so it is constant across evaluation targets by construction; the table therefore reads as "does one domain's profile transfer to another", and the answer is that it does so at $0.417$, no better than guessing. The two domains whose scalars predict a non-positive effect happen to be right about the other non-positive domains, accounting for its partial transfer accuracy.

### A.11 Trajectory length by domain

The admissibility result depends on the distribution of the number of agent decision points $H$, so we report it per domain rather than pooled (Table 10). One *trajectory* here is one logged rollout, so a domain's count is its task count times its trials times its two arms, less any rollout lost to the infrastructure error of Appendix A.1, which is why `telecom`'s count is one short of that product: the $1{,}307$ pooled figure spans every rollout logged in these four two-arm runs, not the $2{,}196$ $\tau^2$ or $6{,}396$ corpus rollouts of Table 3, which add the model-pool and retry-temperature runs and, in the corpus row, BFCL; and `telecom`'s share of it spans all $74$ tasks as launched rather than the $73$

| estimated on | evaluated on | predicted | measured | sign |
|---|---|---|---|---|
| retail train | telecom train | $+0.0315$ | $-0.0913$ | $\times$ |
| retail train | airline train | $+0.0315$ | $+0.0000$ | $\times$ |
| retail train | retail test | $+0.0315$ | $-0.0042$ | $\times$ |
| telecom train | retail train | $-0.0913$ | $+0.0315$ | $\times$ |
| telecom train | airline train | $-0.0913$ | $+0.0000$ | ✓ |
| telecom train | retail test | $-0.0913$ | $-0.0042$ | ✓ |
| airline train | retail train | $+0.0000$ | $+0.0315$ | ✓ |
| airline train | telecom train | $+0.0000$ | $-0.0913$ | $\times$ |
| airline train | retail test | $+0.0000$ | $-0.0042$ | $\times$ |
| retail test | retail train | $-0.0042$ | $+0.0315$ | $\times$ |
| retail test | telecom train | $-0.0042$ | $-0.0913$ | ✓ |
| retail test | airline train | $-0.0042$ | $+0.0000$ | ✓ |

Table 9: Cross-dataset transfer of the four-scalar sign predictor. *predicted* uses the four scalars from the fitting dataset; *measured* is the routing effect on the evaluation dataset.

| dataset | $n$ traj. | mean $H$ | sd | median | min | max | $\mathbb{E}[2^{-H}]$ |
|---|---|---|---|---|---|---|---|
| retail train | 444 | 12.34 | 3.28 | 12 | 6 | 26 | 7.4e-04 |
| telecom train | 443 | 17.42 | 6.72 | 16 | 6 | 51 | 5.0e-04 |
| airline train | 180 | 11.41 | 4.91 | 11 | 3 | 26 | 7.3e-03 |
| retail test | 240 | 11.88 | 2.81 | 11 | 7 | 24 | 9.7e-04 |

Table 10: Distribution of agent decision points per domain, and the resulting step-level admissibility at $K = 2$.

that survive the task-selection rule of Appendix A.1, because that rule drops tasks from the *reward* analysis and not from the logs whose length we are measuring. No domain has a short enough tail to give step-level importance weighting a usable on-support rate: the minimum $H$ anywhere is 3, in `airline`, giving $2^{-3}$ as the maximum observed trajectory support even at $K = 2$, which is how the 6 trajectories of Section 7 clear the threshold at all. The step-to-session ratio this implies runs $69\times$ on airline to $992\times$, so the pooled $312\times$ is not any domain's figure and a practitioner working in one domain cannot take it.

An alternative evaluation design would require additional environment support: if tool effects were transactional, so that a mid-episode switch could be rolled back and replayed, a step-level target would no longer need the whole trajectory to match (Chen et al., 2026b). On both harnesses as shipped the question does not arise, which is why Section 7 compares the two matched designs instead of proposing a third.

### A.12 Escalation signals

The observable proxy escalates when the episode did not terminate through normal user stop, or the step count, tool-error count, or count of near-duplicate consecutive user messages (Jaccard overlap above $0.5$) reaches a threshold. Thresholds were grid-searched on the training split over $\{12, 14, 16, 18, 20, 25, 50\} \times \{1, 2, 3, \infty\}^2$; the reported operating point is the argmax of $112$ configurations selected and scored on the same split, and should be read as such. The text-feature router's own search space is counted in Appendix A.17, including its margin and prediction-threshold sweeps.

The judge signal is one model call per rollout. The judge receives the rendered transcript (user turns, agent turns, agent tool calls with arguments truncated to $300$ characters, tool responses, each message body truncated to $800$ and the assembled transcript to $14{,}000$; all three cuts and the prompt are in Appendix A.7) and is asked whether the agent completely and correctly fulfilled every request, with explicit instructions that a verbal promise without the corresponding tool call does not count, that partial completion does not count, and that a correctly refused request does. It returns a JSON boolean. Parse failures escalate conservatively; there were none. Because the positive class (a genuine failure) has base rate $0.3851$, we report the majority-class accuracy $0.6149$ next to every judge accuracy: the weakest judge does not beat a constant predictor.

| | one-slot | | rescue rate of strong | | compl. | routing effect | |
|---|---|---|---|---|---|---|---|
| dataset | $p_c$ | $p_s$ | after $c$ fails | after $s$ fails | gap | 1st-slot | rescue |
| $\tau^2$ retail train | 0.613 | 0.685 | 0.535 | 0.329 | $+0.206^*$ | $-0.072$ | $+0.104$ |
| $\tau^2$ telecom train | 0.219 | 0.607 | 0.520 | 0.279 | $+0.241^*$ | $-0.388$ | $+0.297$ |
| $\tau^2$ airline train | 0.567 | 0.644 | 0.462 | 0.344 | $+0.118$ | $-0.078$ | $+0.078$ |
| $\tau^2$ retail test | 0.792 | 0.842 | 0.620 | 0.526 | $+0.094$ | $-0.050$ | $+0.046$ |
| BFCL `multi_turn` | 0.675 | 0.697 | 0.426 | 0.099 | $+0.327^*$ | $-0.022$ | $+0.108$ |

Table 11: The routing effect as first-slot cost plus rescue term, both families. The *compl. gap* column is $\mathbb{E}[s \mid c \text{ fails}] - \mathbb{E}[s \mid s \text{ fails}]$, how much better the strong model recovers the cheap model's failures than its own, and $^*$ marks a gap whose 95% CI excludes zero. That star is about the *descriptive* difference between the two columns to its left: the same subtraction against the second slot's unconditional accuracy, an unconditional reference rather than a difficulty-matched comparator, is negative on all 23 measurements, so the column name is shorthand, not a finding. The last two columns sum exactly to Table 1's routing effect.

| significance reading | cut | sig. pos. | rows above | of those, sig. pos. |
|---|---|---|---|---|
| uncorrected 95% CI | 0.182 | 10 | 12 | 0 |
| Bonferroni | 0.155 | 7 | 16 | 0 |
| Benjamini–Hochberg | 0.182 | 8 | 12 | 0 |

Table 12: The screen's cut under each significance reading the paper computes over the same 23-row family. *sig. pos.* is how many measurements that reading calls significantly positive; *rows above* counts measurements strictly above the resulting cut, and the last column is how many of those the same reading calls significantly positive. The last column is zero by construction because each cut is the maximum rate among that convention's significantly positive measurements.

### A.13 Tables referenced from the main text

The following tables provide the detailed results referenced in the main text. Tables 12 and 13 report the self-rescue boundary under each significance correction and its distribution across benchmark families.

**The identity, term by term.** Table 11 is Section 4's decomposition of the routing effect into a first-slot cost and a rescue term on both families. The model-pool extensions appear in Tables 17 and 15.

**The self-rescue boundary and benchmark family.** We define the cut as the largest second-slot self-rescue rate among significantly positive measurements. Consequently, no significantly positive measurement can lie above it in the data used to define it. Table 12 reports this descriptive boundary under three significance conventions: $0.182$ uncorrected, $0.155$ under Bonferroni, and $0.182$ under Benjamini–Hochberg. The zero count above each cut is a consequence of selection, not an independent validation of a predictive screen.

Table 13 shows that the boundary also separates benchmark families and sample sizes. All 12 rows above it come from $\tau^2$, with task counts from 30 to 74 and median interval half-width $0.0473$. All 10 BFCL rows lie at or below it, with $T \geq 199$ and half-width $0.0362$. BFCL's only arm above the cut, `gpt56`, has self-rescue $0.210$ and one-slot accuracy $0.6097$, ranked 5 of 5. As the least accurate arm, it occupies the second slot in 0 of 10 pairs. The above-cut class is therefore restricted to $\tau^2$ by the pool construction. External validation would need to separate self-rescue from these family and sample-size differences.

**All pairs in the BFCL pool.** The ordering is the identity, not model quality: the two pairs whose second slot is `haiku45` (the one model exactly reproducible here, non-unanimity $0.000$) give the two largest routing effects, while the four whose second slot is `glm5`, the most stochastic at $0.182$, give the four smallest. `glm5` is the *strongest* single model on BFCL and still the worst second slot

| | | | above the cut | | at or below | | median | |
|---|---|---|---|---|---|---|---|---|
| family | rows | cells | $n$ | sig. pos. | $n$ | sig. pos. | CI half-width | $T$ |
| $\tau^2$ | 13 | 7 | 12 | 0 | 1 | 1 | 0.0473 | 30–74 |
| BFCL | 10 | 4 | 0 | 0 | 10 | 9 | 0.0362 | 199–200 |

Table 13: The same cut, crossed with benchmark family. *cells* are (dataset, second-slot model) pairs. The above-cut class is entirely $\tau^2$ and the at-or-below class almost entirely BFCL, so the screen's two classes cannot be separated from family, task count or interval width on this corpus.

| dataset | truth sign self-consist. | rule sign accuracy | always-retry baseline | rule MAE |
|---|---|---|---|---|
| retail train | 0.497 | 0.545 | 0.237 | 0.0600 |
| telecom train | 1.000 | 1.000 | 1.000 | 0.0286 |
| airline train | 0.133 | 0.107 | 0.590 | 0.0989 |
| retail test | 0.300 | 0.247 | 0.665 | 0.0316 |

Table 14: A sign predictor derived from the identity's four scalars, with disjoint estimation and evaluation tasks. *Truth sign self-consist.* is how often two disjoint halves of the same dataset agree on the sign of the measured routing effect. It diagnoses whether the target's sign is stable and is *not* a bound on predictor accuracy; see Section C. *rule MAE* is the mean absolute error of the rule's predicted routing effect against the measured one, on the evaluation half. *always-retry baseline* is not an error but a sign accuracy, on the same scale as *rule sign accuracy*: how often the constant prediction "the routing effect is not positive, so retry" gets the sign right. The rule beats that constant on 1 of 4 datasets, ties on 1 and loses on 2, and it never beats the best constant sign predictor (0.727 macro-averaged), which it only equals on `telecom`.

to route into. Table 15 below is the counterpart to Table 17 below on the other benchmark family: same 5 models, opposite conclusion.

The two `haiku45` rows have a degenerate retry baseline. On this pass `haiku45` is unanimous on every task, so its observed self-rescue rate is zero, and a two-slot retry of it therefore returns exactly its one-slot reward. Their retry baseline is *degenerate*: for those two rows the routing effect and the naive gain are numerically the same quantity, because the control the rest of the paper insists on happens to correct by nothing here. That is not a failure of attempt-matching, which did happen (the control spends two `haiku45` rollouts on different trials), and it does not move the cut of Section 8, which is set by the `glm5` rows at 0.182 and is unchanged with these two rows removed. It is the identity's own limiting case, and represents the limiting case of zero observed self-rescue: the largest *non*-degenerate routing effect in the corpus is $\tau^2$'s `qwen3` → `haiku45`, whose second slot's self-rescue rate is 0.0465 rather than zero.

**The BFCL pool, all columns.** Table 15 is the per-pair table Section 5 quotes from, with every column: both arms' one-slot accuracy, the cross-model rescue rate, and the margin against the pool's best two-slot policy.

**The governing axis, all columns.** Table 16 adds source, task count, capability gap and the 95% interval to Figure 3a in the main text.

**All pairs in the $\tau^2$ model pool, with identity terms.** Table 17.

**Hierarchical bootstrap.** Table 18. The task-only interval is what the main tables report; the hierarchical one additionally resamples the three trial indices with replacement, keeping the ordered-pair averaging so that only the uncertainty source changes. We report two versions with different resampling units. In the *common-shock* version a single resampled index triple is shared by every task, so the trial term is a batch-level shock that does not shrink with $T$; it answers "what if the whole batch were rolled again". In the *nested* version each task draws its own triple, which is the two-level cluster bootstrap. Widening is 1.484× on average under the common shock and 1.194× under the nested draw, and no verdict changes under either on these 4 rows. The same common

| 1st slot | 2nd slot | $p_1$ | $p_2$ | routing | cross-resc. | self-resc. | vs. pool |
|---|---|---|---|---|---|---|---|
| gpt56 | haiku45 | 0.610 | 0.673 | $+0.129^*$ | 0.494 | 0.000 | $-0.029$ |
| qwen3 | haiku45 | 0.655 | 0.673 | $+0.111^*$ | 0.374 | 0.000 | $-0.048$ |
| gpt56 | qwen3 | 0.610 | 0.655 | $+0.083^*$ | 0.466 | 0.155 | $-0.040$ |
| haiku45 | sonnet45 | 0.673 | 0.700 | $+0.082^*$ | 0.426 | 0.101 | $-0.019$ |
| qwen3 | sonnet45 | 0.655 | 0.700 | $+0.074^*$ | 0.434 | 0.101 | $-0.027$ |
| gpt56 | sonnet45 | 0.610 | 0.700 | $+0.064^*$ | 0.472 | 0.101 | $-0.038$ |
| sonnet45 | glm5 | 0.700 | 0.742 | $+0.043^*$ | 0.439 | 0.182 | $+0.000$ |
| gpt56 | glm5 | 0.610 | 0.742 | $+0.034^*$ | 0.545 | 0.182 | $-0.009$ |
| qwen3 | glm5 | 0.655 | 0.742 | $+0.030^*$ | 0.476 | 0.182 | $-0.013$ |
| haiku45 | glm5 | 0.673 | 0.742 | $+0.017$ | 0.405 | 0.182 | $-0.026$ |

Table 15: All pairs over the 5 models on BFCL at temperature 0 (except gpt56, whose endpoint would not accept the setting), 199 common tasks, 1 dropped for an inference error in some cell. *routing* is the best cross-model cascade minus retrying the pair's better member *by one-slot accuracy*; $^*$ marks a 95% CI excluding zero. *vs. pool* is against cascade(sonnet45→glm5) $= 0.8317$.

| point | src | $T$ | **self** [95% CI] | cross | gap | routing | 95% CI |
|---|---|---|---|---|---|---|---|
| gpt56→haiku45 | BFCL pool | 199 | 0.000 [0.000,0.000] | 0.494 | 0.064 | $+0.129^*$ | [+0.087,+0.172] |
| qwen3→haiku45 | BFCL pool | 199 | 0.000 [0.000,0.000] | 0.374 | 0.018 | $+0.111^*$ | [+0.072,+0.152] |
| qwen3→haiku45 | $\tau^2$ pool | 74 | 0.046 [0.010,0.101] | 0.475 | 0.059 | $+0.135^*$ | [+0.065,+0.212] |
| two-arm | BFCL | 200 | 0.099 [0.058,0.149] | 0.426 | 0.022 | $+0.087^*$ | [+0.045,+0.130] |
| qwen3→sonnet45 | BFCL pool | 199 | 0.101 [0.059,0.152] | 0.434 | 0.045 | $+0.074^*$ | [+0.039,+0.114] |
| gpt56→sonnet45 | BFCL pool | 199 | 0.101 [0.059,0.152] | 0.472 | 0.090 | $+0.064^*$ | [+0.030,+0.101] |
| gpt56→qwen3 | BFCL pool | 199 | 0.155 [0.107,0.214] | 0.466 | 0.045 | $+0.083^*$ | [+0.047,+0.122] |
| sonnet45→glm5 | BFCL pool | 199 | 0.182 [0.122,0.258] | 0.439 | 0.042 | $+0.043^*$ | [+0.011,+0.079] |
| gpt56→glm5 | BFCL pool | 199 | 0.182 [0.122,0.258] | 0.545 | 0.132 | $+0.034^*$ | [+0.001,+0.068] |
| qwen3→glm5 | BFCL pool | 199 | 0.182 [0.122,0.258] | 0.476 | 0.087 | $+0.030^*$ | [+0.001,+0.063] |
| haiku45→glm5 | BFCL pool | 199 | 0.182 [0.122,0.258] | 0.405 | 0.069 | $+0.017$ | [−0.010,+0.047] |
| telecom train | $\tau^2$ | 73 | 0.279 [0.186,0.392] | 0.520 | 0.388 | $-0.091^*$ | [−0.128,−0.055] |
| sonnet45→glm5 | $\tau^2$ pool | 74 | 0.290 [0.182,0.431] | 0.457 | 0.004 | $+0.050$ | [−0.004,+0.113] |
| haiku45→glm5 | $\tau^2$ pool | 74 | 0.290 [0.182,0.431] | 0.494 | 0.077 | $+0.025$ | [−0.027,+0.086] |
| qwen3→glm5 | $\tau^2$ pool | 74 | 0.290 [0.182,0.431] | 0.500 | 0.135 | $-0.002$ | [−0.036,+0.032] |
| retail train | $\tau^2$ | 74 | 0.329 [0.216,0.475] | 0.535 | 0.072 | $+0.032$ | [−0.036,+0.108] |
| qwen3→sonnet45 | $\tau^2$ pool | 74 | 0.329 [0.217,0.477] | 0.535 | 0.131 | $+0.004$ | [−0.036,+0.047] |
| airline train | $\tau^2$ | 30 | 0.344 [0.191,0.545] | 0.462 | 0.078 | $+0.000$ | [−0.111,+0.122] |
| glm5→gpt56 | $\tau^2$ pool | 74 | 0.407 [0.275,0.565] | 0.507 | 0.068 | $-0.009$ | [−0.054,+0.041] |
| sonnet45→gpt56 | $\tau^2$ pool | 74 | 0.407 [0.275,0.565] | 0.479 | 0.072 | $-0.020$ | [−0.056,+0.020] |
| haiku45→gpt56 | $\tau^2$ pool | 74 | 0.407 [0.275,0.565] | 0.570 | 0.144 | $-0.022$ | [−0.072,+0.032] |
| qwen3→gpt56 | $\tau^2$ pool | 74 | 0.407 [0.275,0.565] | 0.621 | 0.203 | $-0.025$ | [−0.068,+0.020] |
| retail test | $\tau^2$ | 40 | 0.526 [0.296,0.857] | 0.620 | 0.050 | $-0.004$ | [−0.042,+0.033] |

Table 16: All 23 data points behind Figure 3a. *self* is the second-slot model's self-rescue rate; *cross* is $\mathbb{E}[s \mid c \text{ fails}]$; *gap* is the capability gap $p_2 - p_1$. $^*$ marks a 95% CI excluding zero.

shock over the pool pairs widens BFCL's intervals by $1.101\times$ and $\tau^2$'s by $1.358\times$; $\tau^2$'s 1 significant pair survives, and BFCL's 9 becomes 7 under both readings, the rows lost being gpt56→glm5 and qwen3→glm5, the same rows Benjamini–Hochberg drops in Table 19. **Table 18's fourth column is the common-shock version**, the more conservative of the two; the nested version appears here only through its widening factor. The main text's tables report the task-only interval, which is the narrower of the three and the one Table 1's caption names. Excluding degenerate pairs changes the resampling distribution: the index triple is redrawn when all three coincide, so its distribution is conditioned on at least two being distinct (rejection probability $1/9$ at $n = 3$), and a triple such as $(a, a, b)$ gives the surviving pairs $(a, b)$ and $(b, a)$ weight two rather than one. That is forced by the estimator, which is undefined without two independent rollouts. A pair is admitted only when the two resampled indices are *distinct*: a $(i, i)$ pair silently converts the retry baseline into a single strong rollout, and its contribution $\mathbb{E}[(1-s_i)\mathbf{1}\{c_i\}]$ is non-negative by construction, so admitting such pairs

| 1st slot | 2nd slot | $p_1$ | $p_2$ | routing | cross-resc. | self-resc. | vs. pool |
|---|---|---|---|---|---|---|---|
| qwen3 | haiku45 | 0.554 | 0.613 | $+0.135^*$ | 0.475 | 0.046 | $-0.090$ |
| sonnet45 | glm5 | 0.685 | 0.689 | $+0.050$ | 0.457 | 0.290 | $-0.027$ |
| haiku45 | sonnet45 | 0.613 | 0.685 | $+0.032$ | 0.535 | 0.329 | $-0.036$ |
| haiku45 | glm5 | 0.613 | 0.689 | $+0.025$ | 0.494 | 0.290 | $-0.052$ |
| qwen3 | sonnet45 | 0.554 | 0.685 | $+0.004$ | 0.535 | 0.329 | $-0.063$ |
| qwen3 | glm5 | 0.554 | 0.689 | $-0.002$ | 0.500 | 0.290 | $-0.079$ |
| glm5 | gpt56 | 0.689 | 0.757 | $-0.009$ | 0.507 | 0.407 | $-0.009$ |
| sonnet45 | gpt56 | 0.685 | 0.757 | $-0.020$ | 0.479 | 0.407 | $-0.020$ |
| haiku45 | gpt56 | 0.613 | 0.757 | $-0.022$ | 0.570 | 0.407 | $-0.023$ |
| qwen3 | gpt56 | 0.554 | 0.757 | $-0.025$ | 0.621 | 0.407 | $-0.025$ |

Table 17: All pairs over 5 models from 4 vendors, `retail` training split, 74 common tasks. *routing* is against retrying the better model *of that pair*; * marks a 95% CI excluding zero. *vs. pool* is against the best two-slot policy available anywhere in the pool, which is `retry(gpt56)` $= 0.8559$. No pair is shown to beat it; the closest falls short by $-0.0091$ with CI $[-0.0563, +0.0000]$.

| dataset | routing | task-only CI | common-shock CI | widen | bias | flips |
|---|---|---|---|---|---|---|
| retail train | $+0.032$ | $[-0.036, +0.108]$ | $[-0.061, +0.149]$ | $1.45\times$ | $+0.0003$ | no |
| telecom train | $-0.091$ | $[-0.128, -0.055]$ | $[-0.151, -0.027]$ | $1.69\times$ | $+0.0001$ | no |
| airline train | $+0.000$ | $[-0.111, +0.122]$ | $[-0.117, +0.133]$ | $1.07\times$ | $+0.0005$ | no |
| retail test | $-0.004$ | $[-0.042, +0.033]$ | $[-0.054, +0.075]$ | $1.72\times$ | $-0.0002$ | no |

Table 18: Intervals with trials treated as fixed against trials resampled as a batch-level common shock. *common-shock CI* is that second resampling, and it is the only hierarchical column printed; the nested per-task draw is reported in the prose above as a widening factor only, because it is never wider (identical on `airline`) and changes no verdict. *widen* is the ratio of interval widths; *bias* is the replicate mean minus the point estimate, a diagnostic checked by the analysis pipeline. *flips* records whether the significance verdict, the interval excluding zero, changes between the two intervals.

biases the effect upward instead of adding uncertainty. We report the bootstrap bias (replicate mean minus point estimate) per row for exactly this reason; the largest is $0.0005$. The implemented check excludes these degenerate pairs.

**Multiplicity correction over all routing intervals.** Table 19. What the cluster-level permutation test does and does not test is in Appendix A.6, beside the two calculations that put a number on it.

Two-sided bootstrap $p$ values with $20{,}000$ paired task resamples for every row: the pool pairs and BFCL's two-arm result are re-bootstrapped from the logged rollouts rather than approximated from their intervals, which matters because four of these distributions are visibly skewed (`retail`'s hierarchical interval runs $-0.092$ to $+0.117$ around its point estimate), and a Wald $p$ would assume a symmetry they do not have. Each side of the two-sided $p$ carries the $(n + 1)$ correction, so the smallest reportable value is $2/(B{+}1) = 1.0 \times 10^{-4}$, not zero. The 6 rows that reach it print $\leq 10^{-4}$ rather than a value, and their Bonferroni and BH columns inherit that bound because both adjustments are monotone in the raw $p$; draws landing exactly on zero are counted on both sides, the conservative choice for a $1/T$ lattice statistic. Bonferroni and Benjamini–Hochberg run over 23 comparisons, and Bonferroni-adjusted percentile intervals are given for all 23 rows. 8 of 23 survive Bonferroni and 9 survive BH; of the 10 BFCL pool pairs, 5 survive Bonferroni and 6 survive BH. The family is exactly the 23 measurements of Table 16: one deduplication rule, applied once, so the number of intervals we report and the number we correct over cannot diverge. The rule is *arm identity*: a comparison is the same comparison when it is between the same two arms on the same dataset, whatever task count survives the drop rules. It fires once per family. On $\tau^2$ `retail train` the pool pair `haiku45` → `sonnet45` is that dataset's two-arm row on the same tasks, and the two agree to four decimals. On BFCL the same pair is that family's two-arm row on the same rollouts, scored on 200 tasks against 199 because the pool drops one task listwise for an inference error in a fifth arm, so the two read $+0.0867$ and $+0.0821$ and do *not* agree to four decimals. Deduplicating by arm identity avoids counting the BFCL comparison twice despite its

| data point | $T$ | routing | $p$ | Bonferroni | BH | Bonferroni CI |
|---|---|---|---|---|---|---|
| $\tau^2$ telecom train | 73 | $-0.091$ | $\leq 10^{-4}$ | $\leq 0.0023$ | $\leq 0.0004$ | $[-0.151, -0.036]$ |
| BFCL `multi_turn` (T=0) | 200 | $+0.087$ | $\leq 10^{-4}$ | $\leq 0.0023$ | $\leq 0.0004$ | $[+0.027, +0.158]$ |
| BFCL: gpt56 → haiku45 | 199 | $+0.129$ | $\leq 10^{-4}$ | $\leq 0.0023$ | $\leq 0.0004$ | $[+0.067, +0.203]$ |
| BFCL: qwen3 → haiku45 | 199 | $+0.111$ | $\leq 10^{-4}$ | $\leq 0.0023$ | $\leq 0.0004$ | $[+0.052, +0.179]$ |
| BFCL: gpt56 → qwen3 | 199 | $+0.083$ | $\leq 10^{-4}$ | $\leq 0.0023$ | $\leq 0.0004$ | $[+0.029, +0.147]$ |
| BFCL: qwen3 → sonnet45 | 199 | $+0.074$ | $\leq 10^{-4}$ | $\leq 0.0023$ | $\leq 0.0004$ | $[+0.021, +0.139]$ |
| $\tau^2$: qwen3 → haiku45 | 74 | $+0.135$ | 0.0002 | 0.0046 | 0.0006 | $[+0.034, +0.261]$ |
| BFCL: gpt56 → sonnet45 | 199 | $+0.064$ | 0.0002 | 0.0046 | 0.0006 | $[+0.013, +0.122]$ |
| BFCL: sonnet45 → glm5 | 199 | $+0.043$ | 0.0072 | 0.1656 | 0.0184 | $[-0.004, +0.101]$ |
| BFCL: gpt56 → glm5 | 199 | $+0.034$ | 0.0377 | 0.8670 | 0.0867 | $[-0.013, +0.089]$ |
| BFCL: qwen3 → glm5 | 199 | $+0.030$ | 0.0471 | 1.0000 | 0.0985 | $[-0.012, +0.084]$ |
| $\tau^2$: sonnet45 → glm5 | 74 | $+0.050$ | 0.0888 | 1.0000 | 0.1702 | $[-0.032, +0.153]$ |
| $\tau^2$: qwen3 → gpt56 | 74 | $-0.025$ | 0.2786 | 1.0000 | 0.4636 | $[-0.092, +0.045]$ |
| BFCL: haiku45 → glm5 | 199 | $+0.017$ | 0.2822 | 1.0000 | 0.4636 | $[-0.025, +0.067]$ |
| $\tau^2$: sonnet45 → gpt56 | 74 | $-0.020$ | 0.3213 | 1.0000 | 0.4926 | $[-0.077, +0.050]$ |
| $\tau^2$: haiku45 → gpt56 | 74 | $-0.022$ | 0.4029 | 1.0000 | 0.5485 | $[-0.097, +0.070]$ |
| $\tau^2$: haiku45 → glm5 | 74 | $+0.025$ | 0.4264 | 1.0000 | 0.5485 | $[-0.052, +0.122]$ |
| $\tau^2$ retail train | 74 | $+0.032$ | 0.4293 | 1.0000 | 0.5485 | $[-0.068, +0.158]$ |
| $\tau^2$: glm5 → gpt56 | 74 | $-0.009$ | 0.7163 | 1.0000 | 0.8671 | $[-0.077, +0.072]$ |
| $\tau^2$: qwen3 → sonnet45 | 74 | $+0.004$ | 0.9031 | 1.0000 | 0.9939 | $[-0.059, +0.077]$ |
| $\tau^2$ retail test | 40 | $-0.004$ | 0.9147 | 1.0000 | 0.9939 | $[-0.067, +0.054]$ |
| $\tau^2$: qwen3 → glm5 | 74 | $-0.002$ | 0.9507 | 1.0000 | 0.9939 | $[-0.052, +0.052]$ |
| $\tau^2$ airline train | 30 | $+0.000$ | 1.0000 | 1.0000 | 1.0000 | $[-0.156, +0.200]$ |

Table 19: All 23 routing intervals with multiplicity control. 8 survive Bonferroni (2 two-arm results and 6 pool pairs, including `telecom` and BFCL's two-arm result, both at the adjusted floor 0.0023) and 9 survive Benjamini–Hochberg. This caption's counts are generated from the same file as the table.

| $K$ | step-level $\mathbb{E}[K^{-H}]$ | session-level $1/K$ | ratio | logs per usable traj. |
|---|---|---|---|---|
| 2 | 7.41e-04 | 0.500 | 674× | 1,349 |
| 3 | 2.31e-05 | 0.333 | 14412× | 43,237 |
| 4 | 2.55e-06 | 0.250 | 98157× | 392,630 |
| 6 | 1.49e-07 | 0.167 | 1118330× | 6,709,977 |

Table 20: Admissibility for a deterministic target under uniform behaviour, using the measured distribution of $H$ over the 444 trajectories of `retail` train *only* – not the 1,307 pooled across all four datasets that Section 7 counts, which is why 0.0% here and the 6 trajectories there are consistent: the survivors are in `airline`, whose minimum $H$ is 3 (Appendix A.11). Reaching 5% at $K$=2 requires $H \leq 4$.

different retained task counts. BFCL's unpinned-temperature row in Table 1 is a third thing again, a second configuration of the same 200 tasks rather than a further comparison; adding it leaves the Bonferroni survivors unchanged and adds only itself under BH.

**The decomposition as a figure.** Figure 1 ❶ in the main text plots Table 1 per dataset, with all three intervals rather than the routing term's alone. The admissibility curves are Figure 4b.

**Admissibility by $K$.** Table 20, discussed in Section 7.

**Session-level off-policy evaluation per log.** We build seven candidate session-level routers, compute their true values from the full trial matrix, and estimate each from logged datasets formed by drawing one arm per task uniformly. Table 21 reports, per dataset, the maximum absolute bias over logs, the rank correlation of the average over logs, and the distribution of the per-log rank correlation.

| | | Spearman vs. truth | | | | selection regret | |
|---|---|---|---|---|---|---|---|
| dataset | max bias | mean-of-logs | median | 5th pct | $\geq 0.7$ | mean | $> 0$ |
| retail train | 0.0145 | 0.964 | 0.486 | −0.150 | 20% | 0.047 | 46% |
| telecom train | 0.0126 | 1.000 | 0.937 | 0.595 | 90% | 0.019 | 17% |
| airline train | 0.0117 | 0.991 | 0.386 | −0.109 | 10% | 0.042 | 78% |
| retail test | 0.0205 | 0.982 | 0.329 | −0.193 | 8% | 0.035 | 75% |

Table 21: Session-level off-policy evaluation, per logged dataset. The estimator is unbiased (*max bias*), and the rank correlation of the average over logs (*mean-of-logs*) is high, but the distribution over single logs is not. *selection regret* is the true reward of the candidate the estimator ranks first, subtracted from the true reward of the actually best candidate, so it is 0 when the estimator picks the right policy and positive otherwise; we report its mean over logs and the fraction of logs on which it is positive. Ranking is reliable only where candidate values are well separated.

| judge | acc. [95% CI] | recall | prec. | missed | wasted | cascade gain [95% CI] |
|---|---|---|---|---|---|---|
| `haiku45` | 0.574 [0.47,0.68] | 0.772 | 0.468 | 13 | 50 | +0.0135 [−0.041,+0.068] |
| `sonnet45` | 0.682 [0.58,0.78] | 0.807 | 0.561 | 11 | 36 | +0.0135 [−0.047,+0.074] |
| `opus45` | 0.595 [0.49,0.70] | 0.509 | 0.475 | 28 | 32 | −0.0068 [−0.074,+0.061] |

Table 22: The escalation signal as a failure detector. Accuracy does not consistently increase with vendor tier; the intervals do not resolve an ordering. Against a 0.3851 base rate for genuine failure, never-escalating scores 0.6149 and always-escalating 0.3851; 2 of 3 judges fall below the former. *missed* counts rollouts the benchmark verifier scored as failures and the judge passed, the error that costs quality; *wasted* counts rollouts the verifier scored as successes and the judge escalated, the error that costs money. Gains are against one-slot always-strong, the unequal-budget comparison Section 3 rejects; against the two-slot retry control every judge-gated policy is negative.

**The escalation judge as a failure detector.** Table 22, plotted as Figure 4a in the main text.

**Every trial, unaveraged.** Table 23 gives the one-slot accuracy of every (dataset, arm, trial) cell in the corpus, with no averaging, because every interval in this paper rests on the three repeats and a reader should be able to see them. Two things are visible directly. No arm mean is carried by a single favourable trial: the largest within-arm spread anywhere is 0.100, on the 30-task `airline` split. And the strong arm's spread exceeds the cheap arm's on 4 of 4 $\tau^2$ datasets: the per-arm noise asymmetry of Appendix B in raw form, before it is turned into a noise floor or a self-rescue rate.

**Per-arm noise floors.** Table 24, discussed in Appendix B.

**The same model on both benchmark families.** Table 25, discussed in Section 5. Generated from raw rollouts on the pinned BFCL pass.

### A.14 The one-call per-task oracle, and why it does not bound the same thing

Section 3's residual bounds two-slot routers. It says nothing about a router that makes *one* call per task, and the two are not the same quantity: the retry confound exists because a cascade spends two attempts, so a policy that spends one cannot inherit it.

Table 27 gives the one-call per-task oracle, $\mathbb{E}_t[\max_k \bar{r}_k(t)]$ against the best single arm, in both of the two forms this paper can compute, because they do not support the same conclusion. Read with the arm selected on the rollouts that score it, it recovers 95% to 100% of the two-slot naive gain on three of the four $\tau^2$ datasets with no second attempt. That reading is an upper bound, and by our own argument it is not a small one: choosing the arm on the trials used to score it inflates the gain (Appendix B). Corrected by the split-half rule – the arm is chosen on one subset of trials and scored on the disjoint rest, averaged over every split – the gain is 25% to 82% of the two-slot naive gain, and its interval excludes zero on 1 of the 4 datasets: `retail` train at $+0.1014$ $[+0.0248, +0.1509]$. On the other three the corrected interval covers zero, widest on `airline`,

| dataset | arm | $T$ | trial 1 | trial 2 | trial 3 | spread |
|---|---|---|---|---|---|---|
| retail train | cheap | 74 | 0.608 | 0.622 | 0.608 | 0.014 |
| retail train | strong | 74 | 0.703 | 0.703 | 0.649 | 0.054 |
| telecom train | cheap | 73 | 0.219 | 0.233 | 0.205 | 0.027 |
| telecom train | strong | 73 | 0.575 | 0.603 | 0.644 | 0.069 |
| airline train | cheap | 30 | 0.567 | 0.567 | 0.567 | 0.000 |
| airline train | strong | 30 | 0.633 | 0.700 | 0.600 | 0.100 |
| retail test | cheap | 40 | 0.800 | 0.800 | 0.775 | 0.025 |
| retail test | strong | 40 | 0.800 | 0.875 | 0.850 | 0.075 |
| BFCL | `glm5` | 199 | 0.754 | 0.739 | 0.734 | 0.020 |
| BFCL | `gpt56` | 199 | 0.603 | 0.608 | 0.618 | 0.015 |
| BFCL | `haiku45` | 199 | 0.673 | 0.673 | 0.673 | 0.000 |
| BFCL | `qwen3` | 199 | 0.663 | 0.643 | 0.658 | 0.020 |
| BFCL | `sonnet45` | 199 | 0.683 | 0.714 | 0.704 | 0.030 |

Table 23: One-slot accuracy per trial, unaveraged. *spread* is the maximum minus the minimum across that arm's trials. $\tau^2$ rows use each dataset's two arms; BFCL rows use the 5-model pool. `haiku45` on BFCL is exactly reproducible across all three trials, which eliminates the observed gain from retrying that model.

| | mean reward | | noise floor | | | crossing |
|---|---|---|---|---|---|---|
| dataset | cheap | strong | cheap | strong | ratio | tasks |
| retail train | 0.613 | 0.685 | 0.054 | 0.311 | 5.7 | 0 |
| telecom train | 0.219 | 0.607 | 0.055 | 0.329 | 6.0 | 1 |
| airline train | 0.567 | 0.644 | 0.000 | 0.367 | $\infty$ | 0 |
| retail test | 0.792 | 0.842 | 0.025 | 0.250 | 10.0 | 0 |

Table 24: Per-arm noise floors. *crossing tasks* counts tasks on which the two arms trade wins across trials, the condition under which a per-task-maximum oracle is inflated.

where $T = 30$ turns a reused-selection $+0.1223$ into $+0.1000$ $[-0.0111, +0.1667]$, consistent with the failed pre-registered arm-selection criterion in Appendix C. A second inflation, trial noise, is separately absent on three datasets: by the criterion of Appendix B, per-task maxima are inflated by noise only when arms trade wins across trials, and those three have 0 such tasks, so the reused-selection oracle is numerically identical whether computed from one trial or three (*stable* column). `telecom` is the exception with one crossing task, and it is the dataset where the one-call gain is smallest either way ($+0.0228$ reused, $+0.0046$ corrected).

The cascade comparison does not bound a one-call dispatcher's attainable gain. The corrected oracle establishes positive one-call headroom on `retail` train and leaves it unresolved on the other $\tau^2$ datasets. On the held-out split, correction reduces the estimate from $+0.0750$ to $+0.0250$. Section 6 evaluates a particular lexical dispatcher, which fails to capture the corrected headroom; this does not rule out other dispatchers.

Three headroom estimates for the held-out `retail` split use different estimators: $+0.0750$ is this table's reused-selection row: the per-task maximum over the two arms' three-trial mean rewards, minus the better arm, over every test task. $+0.0250$ $[-0.0083, +0.1000]$ is that row's selection-corrected column, and it is the one the abstract's scope limit is counted against. $+0.0500$ is Section 6's *honest-oracle* headroom, which is the quantity a router could actually have captured under that section's nested split: the arm is chosen on a label subset of trials and scored on a disjoint subset, so the selection cannot use the rollout it is scored on. The first two differ by the selection correction. The second and third differ by their scoring *basis* and not by that reuse at all, because the correction depends on which trials are averaged. Averaged over every single-trial basis, as this table's row does, the correction is worth $+0.0250$ against $+0.0750$; scored on Section 6's single held-out trial, the honest and the reuse oracle coincide exactly, so on *that* basis reuse costs $+0.0000$. These estimates answer different questions. On the held-out trial the oracle is $0.9000$ and the better arm scores $0.8500$. The router is compared with this pair because it uses the same scoring basis.

| | $\tau^2$ retail | | BFCL (pinned $T$) | | |
|---|---|---|---|---|---|
| model | acc. | self-rescue | acc. | self-rescue | $\Delta$ self |
| glm5 | 0.689 | 0.290 | 0.742 | 0.182 | $-0.108$ |
| gpt56 | 0.757 | 0.407 | 0.610 | 0.210 | $-0.197$ |
| haiku45 | 0.613 | 0.046 | 0.673 | 0.000 | $-0.046$ |
| qwen3 | 0.554 | 0.323 | 0.655 | 0.155 | $-0.168$ |
| sonnet45 | 0.685 | 0.329 | 0.700 | 0.101 | $-0.228$ |

Table 25: Accuracy and self-rescue rate for each of the 5 models we ran on both families. $\Delta$ self is BFCL minus $\tau^2$: it is negative for all 5 models, so the benchmark moves every model in the same direction. Within BFCL the rate still spans $0.2103$, so it is not a benchmark constant either.

| | | selection reused | | | selection corrected | | label |
|---|---|---|---|---|---|---|---|
| dataset | $K$ | oracle | best arm | gain | gain | 95% CI | ties |
| $\tau^2$ retail train | 2 | 0.8153 | 0.6847 | $+0.1306$ | $+0.1014$ | $[+0.0270, +0.1509]$ | 61% |
| $\tau^2$ retail train | 5 | 0.9099 | 0.7568 | $+0.1532$ | $+0.0327$ | $[-0.0135, +0.0833]$ | 84% |
| BFCL multi_turn | 2 | 0.8124 | 0.7002 | $+0.1122$ | $+0.1005$ | $[+0.0662, +0.1382]$ | 73% |
| BFCL multi_turn | 5 | 0.8995 | 0.7420 | $+0.1575$ | $+0.0603$ | $[+0.0109, +0.0737]$ | 88% |

Table 26: The one-call per-task oracle at two arms and over the full 5-model pool, on both benchmark families. *ties* is the fraction of tasks on which the label trials leave the arm argmax tied, resolved to the globally best single arm. The two-arm $\tau^2$ row reproduces the published two-arm numbers, which the generator checks before writing the pool rows.

**The same oracle over a pool, and on BFCL.** A one-call dispatch router dispatches over a pool, and every row above is two arms on $\tau^2$, so Table 26 recomputes the oracle at both scales on both families. The effects of adding BFCL and adding models differ. BFCL's two-arm row *adds* a dataset whose corrected interval excludes zero, at $+0.1005$ $[+0.0662, +0.1382]$, which is why the count this paper quotes is 2 of 5 rather than of the four $\tau^2$ datasets alone. For the pool rows, adding arms raises the oracle level, as it must, but it raises the label-tie rate more. On $\tau^2$ retail train the reused-selection oracle rises from $0.8153$ to $0.9099$ while the corrected gain falls from $+0.1014$ to $+0.0327$ $[-0.0135, +0.0833]$, which covers zero, and the fraction of tasks where the argmax over arms is a tie on the label trials goes from $61\%$ to $84\%$. With five arms and one or two binary label trials, most tasks carry no information about which arm to pick, and the tie-break (the globally best single arm, never list order) is doing the work. At this trial count, the measured selection-corrected headroom shrinks as the pool grows. We therefore distinguish the two-arm estimates from the pool estimates.

| | selection reused | | | selection corrected | | | |
|---|---|---|---|---|---|---|---|
| dataset | oracle | best arm | gain | gain | 95% CI | cross | stable |
| retail train | 0.8153 | 0.6847 | $+0.1306$ | $+0.1014$ | $[+0.0248, +0.1509]$ | 0 | yes |
| telecom train | 0.6301 | 0.6073 | $+0.0228$ | $+0.0046$ | $[-0.0274, +0.0457]$ | 1 | no |
| airline train | 0.7667 | 0.6444 | $+0.1223$ | $+0.1000$ | $[-0.0111, +0.1667]$ | 0 | yes |
| retail test | 0.9167 | 0.8417 | $+0.0750$ | $+0.0250$ | $[-0.0083, +0.1000]$ | 0 | yes |

Table 27: The one-call per-task oracle. *oracle* is $\mathbb{E}_t[\max_k \bar{r}_k(t)]$ over the two arms of Table 1, *best arm* the better single arm on the same tasks, and *gain* their difference, all three with the arm selected on the same rollouts that score it. The *selection corrected* pair repeats the gain with the arm chosen on a subset of trials and scored on the disjoint rest, averaged over every split, with a task-bootstrap $95\%$ interval. *cross* counts tasks where the arms trade wins across trials, the condition under which a per-task maximum is inflated (Appendix B); *stable* records whether the oracle is unchanged when computed from one trial instead of three. Compare *gain* against the *naive gain* column of Table 1, which buys its gain with a second attempt.

| dataset | $T$ | retry×3 | best cross | routing effect | 95% CI | 1-sided $p$ |
|---|---|---|---|---|---|---|
| retail train | 74 | 0.824 | 0.883 | +0.059 | [+0.000, +0.126] | 0.028 |
| telecom train | 73 | 0.781 | 0.722 | −0.059 | [−0.091, −0.032] | 1.000 |
| airline train | 30 | 0.833 | 0.856 | +0.022 | [−0.067, +0.133] | 0.408 |
| retail test | 40 | 0.950 | 0.950 | +0.000 | [+0.000, +0.000] | 1.000 |

Table 28: Three-slot budget. *retry*×3 is the strong model in all three slots (`sss`), the baseline; *best cross* is the maximum over the four mixed orderings `css`, `ccs`, `scs`, `ssc`, which biases the comparison in favour of routing. The routing effect nonetheless does not clear zero after a multiplicity correction on `retail` and is negative on `telecom`.

### A.15 The three-slot budget in full

Section 3 reports the three-slot result in one sentence; Table 28 is the full comparison. The budget enumerates all $3! = 6$ ordered triples of trials, verifier-gated at each slot. *best cross* is the maximum over four of the six mixed slot orderings, written cheap/ strong per slot: `css`, `ccs`, `scs` and `ssc`. The two omitted, `csc` and `scc`, are the two that spend two of three slots on the cheap model and still end on it; `ssc` also ends on the cheap model and is kept, so this subset is not motivated by a distinct policy principle. The reported maximum is unchanged: we compute all six and the generator refuses to build unless the maximum over these four equals the maximum over all six, on all $4$ datasets. The baseline is `sss`, the strong model three times; `ccc` is computed and not used, because a cheap-only baseline would flatter routing. Taking a maximum over four orderings against one fixed baseline is generous to routing, deliberately. On `retail` the routing effect *grows* to $+0.0586$ (one-sided bootstrap $p = 0.028$), positive at the conventional level, even though its percentile interval $[+0.0000, +0.1261]$ rests on zero: the statistic is a multiple of $1/T$ and the bootstrap places an atom there. We treat intervals with either endpoint at zero as covering zero throughout, including Appendix A.20. The one-sided test provides additional information here. Applying Bonferroni across the four three-slot datasets puts the `retail` result above $0.05$, so the increase is not significant after correction. On `telecom` the effect stays significantly negative at $-0.0594$ $[-0.0913, -0.0320]$; `airline` at $+0.0222$ keeps an interval that covers zero, and on the held-out split the two policies agree on every task ($+0.0000$, hence a degenerate interval). The identity says why the sign can move this way: a third slot saturates the retry baseline faster than it saturates the cross-model one, so the residual between them can widen with budget even though each arm individually saturates.

### A.16 Why the identity's four scalars do not yield a usable sign predictor

Four cheap scalars determine the routing effect, which suggests profiling a small sample of tasks and predicting the sign before committing to a policy. The rule fails on $\tau^2$: macro sign accuracy $0.475$ against $0.727$ for the best constant predictor, with disjoint estimation and evaluation tasks over $400$ random splits per dataset. The target sign is unstable: disjoint halves of the same dataset agree only $0.482$ of the time. This diagnoses variability but does not upper-bound another predictor's accuracy (Appendix C). The exception is `telecom`, where the capability gap is large enough that the first-slot cost dominates and the rule is right $1.000$ of the time. Tables 14, 8 and 9 give sign accuracy against profiling-sample size and across datasets, and Appendix A.10 reads the size curve and the transfer test row by row.

### A.17 The text-feature router in detail

The router sees only `user_scenario.instructions`: the reason for the call, what the persona knows, what it does not know, and stylistic instructions. It never sees `evaluation_criteria`, which contains gold actions and is unavailable at deployment. Features are (a) hand-built counts: character and word length, line count, order-like identifiers, conjunctions, question marks, type-token ratio, an unknown-information flag, capitalised-token mass, and counts for twenty domain verbs; (b) word 1–2-gram TF-IDF, $\leq 4000$ features, minimum document frequency 2; (c) character 3–5-gram TF-IDF with the same caps. Models: logistic regression at two regularisation strengths, gradient boosting, random forest, and ridge on the signed arm difference. We additionally swept a label margin over $\{0, 0.34, 0.5, 0.67, 1.0\}$ and a prediction threshold over $\{0.5, 0.6, 0.7, 0.8\}$; large margins drive the router to the constant always-strong policy.

Evaluation is nested and 5-fold over tasks: trials are split into a label subset and a scoring subset, and tasks into folds, so no task contributes its label and its score from the same rollout. One configuration is selected on the training split and only that configuration is evaluated on the held-out test split, where it scores $-0.0250$ $[-0.1000, +0.0000]$. Ties in any label construction resolve to the globally better arm, never to list order. With one label trial, many tasks tie, so list-order tie-breaking would introduce an arbitrary training signal. Selecting the best of 11 configurations on the test split would give $+0.0250$; we report this only as an illustration of selection bias. The training search contains 106 distinct configurations: the 11 base (feature set, model) variants, plus a $5 \times 4$ margin-and-threshold grid for 5 variants, with duplicated base settings counted once. The grid was not run on the test split.

**Task-level gains and the BFCL evaluation.** Task counts clarify the scale of the $\tau^2$ result. In tasks rather than reward points, the held-out result is $-1$ against a headroom of 2 out of 40, and the statistic can only move in units of $1/40$; on the training split the corresponding headroom is 10 of 74. And the selected configuration is nearly the constant policy: it departs from the single best arm on 2 of 40 held-out tasks and 9 of 74 training tasks, so the entire held-out loss comes from a handful of decisions. A result that turns on one or two flipped tasks cannot rule out a feature class, which is why we ran the same router on BFCL, where 199 tasks and 5 arms were already logged providing a larger evaluation set. It scores $-0.0201$ $[-0.0553, +0.0000]$ against a corrected headroom of $+0.1307$ ($-4$ tasks against 26), capturing $-15\%$, and it departs from `glm5` on 26 tasks. The sign is the same and the interval is 1.8 times narrower, providing stronger evidence for the tested lexical router: on two independently implemented benchmarks a router seeing only the request text does not beat the best fixed arm, while a selection-corrected oracle over the same arms is worth an order of magnitude more than the router captures. Two limitations affect transfer. The hand-built feature set's verb list is $\tau^2$ vocabulary and has little coverage of BFCL's file-system and vehicle tasks, so the hand-built variants understate what a BFCL-tuned lexical router would do; the TF-IDF and character-$n$-gram variants carry no such vocabulary and land in the same place. And one variant regresses the signed difference between two arms and thresholds it at zero, which cannot express a five-way choice at all; it is reported and flagged rather than dropped.

**Scope of the lexical router result.** The features are counts and word or character $n$-grams. We use no sentence embeddings, pretrained encoder, preference model or language-model assessment of difficulty. The result therefore concerns this feature class on these datasets. A semantic router might capture distinctions that lexical features miss. The held-out headroom, $+0.0500$, is also small relative to the interval $[-0.1000, +0.0000]$, limiting the precision of architecture comparisons. Training a dense router on the same 74 tasks would still face selection uncertainty; Appendix A.21 shows how in-sample threshold selection overstates transferred performance. We leave semantic routing and evaluation on larger task sets to future work.

### A.18 Session-level off-policy evaluation with finite logs

Support of $1/K$ does not by itself make an estimate accurate. Averaged over many logs the session-level estimator is unbiased (maximum absolute bias 0.0205 across the four datasets), but a practitioner has one log. Per log, the rank correlation with truth has median 0.486 on `retail` and reaches 0.7 in only 20% of logs. Reporting instead the rank correlation of the *average* over logs measures unbiasedness rather than usability and reads as 1.000; it is not what anyone obtains from a single log. Table 21 gives the per-dataset figures and the construction.

### A.19 How informative each interval is

On `airline` ($T = 30$) the routing interval $[-0.1111, +0.1222]$ is as wide as the dataset's entire naive gain $+0.1222$, so it permits a retry share anywhere from nothing to more than all of it. `retail`'s upper bound is 80% of its naive gain, and the held-out split's is tighter still at 42%. We report the four intervals separately.

Table 29 is the same point made for every term rather than only the routing one. The body table prints one interval per row because six more columns do not fit; all three are here, and so is the retry share as a resampled ratio rather than as a bare multiplier. The ratio is especially uncertain when its denominator is near zero. The share the main text quotes at its largest, 6.0 times the naive gain on

| benchmark / dataset | naive gain | retry effect | **routing effect** |
|---|---|---|---|
| $\tau^2$ retail train | $+0.135$ [$+0.072$, $+0.207$] | $+0.104$ [$+0.068$, $+0.140$] | $+0.032$ [$-0.036$, $+0.108$] |
| $\tau^2$ telecom train | $+0.018$ [$+0.000$, $+0.041$] | $+0.110$ [$+0.073$, $+0.146$] | $-0.091$ [$-0.128$, $-0.055$] |
| $\tau^2$ airline train | $+0.122$ [$+0.033$, $+0.233$] | $+0.122$ [$+0.067$, $+0.178$] | $+0.000$ [$-0.111$, $+0.122$] |
| $\tau^2$ retail test | $+0.079$ [$+0.025$, $+0.142$] | $+0.083$ [$+0.042$, $+0.133$] | $-0.004$ [$-0.042$, $+0.033$] |
| BFCL `multi_turn` | $+0.117$ [$+0.077$, $+0.160$] | $+0.030$ [$+0.018$, $+0.043$] | $+0.087$ [$+0.045$, $+0.130$] |
| *unpinned T* | $+0.095$ [$+0.064$, $+0.129$] | $+0.048$ [$+0.033$, $+0.065$] | $+0.047$ [$+0.015$, $+0.081$] |

Table 29: All three terms of the decomposition with intervals, on both families, plus retry's share of the naive gain resampled on the same paired task draws. An $\infty$ upper end marks a share whose denominator crosses zero in more than $2.5\%$ of resamples, which makes the upper percentile undefined rather than large.

`telecom`, has a denominator whose own $95\%$ interval starts at zero: $4.9\%$ of the paired resamples give a naive gain of zero or less, and above $2.5\%$ the upper percentile of a ratio is not defined on the resampling distribution at all. So that entry is one-sided, bounded below by $2.6$ with a ninetieth percentile at $16$; what the data support is the direction and the lower bound, not the point. The other rows' denominators are far from zero and their ratios are two-sided.

### A.20 The domination test in full

Against the best two-slot policy anywhere in the $\tau^2$ pool (`retry(gpt56)`, at $0.8559$), the one significantly positive pair falls short by $-0.0901$, and $0$ of $10$ pairs clear it. This does not establish that all pairs are dominated: the pool best is an in-sample maximum over $10$ candidates, and the closest pair falls short by only $-0.0091$ $[-0.0563, +0.0000]$, an interval that covers zero. The interval therefore does not establish strict domination.

Replacing each pair's baseline with a *two-slot* retry of the pool's better member, rather than its one-slot reward (a change that otherwise favours routing precisely on low-self-rescue models), drops the one significant pair to $+0.0676$ $[-0.0045, +0.1486]$, leaving $0$ of $10$ significant.

**The same comparisons on BFCL.** Three different baselines give three counts on BFCL, two of which coincide, and Section 5 names all three, so we put them side by side here. **Both are two-slot retry baselines**; they differ only in how the retried member is picked. Pick it by one-slot accuracy and $9$ of $10$ pairs are significantly positive; pick it by two-slot reward and $9$ are. The two choices pick a different member on two of the ten rows but reach the same significance verdict on all ten, which is why both counts are $9$ and not merely close. Neither is a one-slot comparison: `bfcl_pool.py` forms both as `cascade` minus `two(m, m)`. Against the best two-slot policy anywhere in the pool (`cascade(sonnet45→glm5)`, at $0.8317$), only $1$ of $10$ attains it, and that pair is the pool best itself. The first two counts are large and the third is small because the pool-best baseline is a different and much harder comparison: it lets the baseline draw its second slot from any model in the pool, while the within-pair tests confine it to the pair. The pool-best pair equals its own baseline; this is not a positive margin over that baseline. No other pair is shown to exceed the pool best.

### A.21 Thresholds transfer worse than selecting them in sample suggests

Thresholds fixed on the `retail` training split and transferred unchanged to the held-out split give $0.568\times$ the cost of always-strong at a quality change of $-0.0500$ $[-0.1667, +0.0583]$, an interval covering zero. Selecting and scoring the threshold on the same split would instead report $0.307\times$. The gap between those two numbers is the same selection effect as the honest-oracle correction, one level up: it is why we recommend holding the threshold out rather than reporting the in-sample optimum.

### A.22 Unmeasured properties of the model pool

Section 9 covers what our *measurements* bound. Three properties of the pool itself sit outside the design, and one of them is a cheaper alternative to routing that could reduce the first-slot cost, the larger of the identity's two terms on the datasets where routing does worst.

**The arms are hosted endpoints we cannot inspect.** Every model in this paper is served behind an API. We record the endpoint identifier per run (Table 2) and we can read back the decoding parameters the endpoint accepted, but not the weights, the serving stack, or whether the provider changed either between our passes. Fine-tuning interfaces are one documented way a hosted model's behaviour diverges from the base model a reader might assume it is (Chen et al., 2024); a provider-side change of that kind would move $p_c$ and $p_s$, and therefore the first-slot cost, with no signal in our logs. It could also change self-rescue. The retry effect is the strong arm's non-unanimity divided by the trial count (Appendix A.26), and non-unanimity on a hosted endpoint at temperature $0$ is a property of the serving stack, not of the weights: work on replaying logged trajectories under a switched model reports exactly this quantity swinging from near-total divergence to near-identity with the serving configuration (Gonuguntla, 2026). So a more deterministic serving path would lower the retry effect and raise the routing residual, and the $\tau^2$ null is conditional on this path in a way no measurement here bounds. Section 3's heading and Section 9 say so; the experiment that would settle it is one $\tau^2$ dataset rerun through a second serving path, which we did not do. This is a reproducibility limit on every API-only agent-evaluation paper, ours included, We support reproducibility with the recorded endpoint identifiers, the per-rollout reward matrix and the per-file checksums in the supplementary archive (Appendix A.25).

**We did not test for benchmark contamination.** Both benchmark families are public. If a model's training data contains their tasks, per-task success rates and therefore the whole identity are measured on a partly memorised distribution. Token-level dataset-inference methods can test exactly this from model outputs (Xiong et al., 2026) and we did not run one. The direction of any resulting bias is uncertain: contamination would inflate $p_c$ and $p_s$ together, so its effect on the *difference* the first-slot cost measures is not signed a priori.

**Distillation as an untested alternative.** The identity's first-slot cost is $p_c - p_s$, the term that decides the sign wherever the capability gap is large (`telecom`). A practitioner facing that gap has a second option we never put on the table: raise $p_c$ instead of routing around it, by distilling the strong model into the cheap one (Gu et al., 2023), a move whose applied form reaches domains far from ours (Zhang et al., 2025). Our whole comparison is routing against *retrying*, and on our numbers the first-slot cost is the larger of the identity's two terms on the dataset where routing does worst, so the alternative that shrinks it directly is the one with the most room. We cannot say whether it wins, only that this paper does not test it and that the identity says where it would act.

### A.23 Baselines in the cited routing and cascade studies

Table 30 documents the baselines in the 10 cascade and router papers we cite: each paper's headline comparator, whether model-call counts are matched, and a verbatim sentence from the body or a caption. All entries were checked against the full papers retrieved on the same day.

Of these papers, 0 report a same-model retry or repeated-sampling control at matched attempts. 4 equalise spend without matching call counts, 2 equalise neither, and 1 includes a call-count axis in its headline plot. Four compare a multi-call policy with a single-call baseline; only one of those four charges the extra calls. However, 3 of the 10 are one-model-per-request dispatchers, whose attempt budgets already match. Our critique therefore applies to the cascade comparisons in this audit, not to routing literature as a whole. Two papers fall outside the comparison: one reports routing accuracy without cost, and one routes context rather than models.

### A.24 Extended related work: what else moves the two terms of the identity

**Why step-level replay fails.** Gonuguntla (2026) showed that replaying logged trajectories under a switched model scores the wrong world and Ganz et al. (2026) measured the cost of continuing another model's trajectory; we study the related prerequisite of how often logged trajectories support a target policy. The following work addresses related evaluation choices.

The main text's related work positions this paper against routers, against work that treats the attempt budget as the treatment, and against the harness-repair literature whose domain-dependence result matches ours. Step-level replay, and the rest of that literature, are here. This section covers work

| paper | headline baseline | attempts held fixed | retry ctrl. | where |
|---|---|---|---|---|
| Zhou et al. (2026b) | always-one-model rows as a performance floor, per-task oracle as ceiling<br>*figure caption:* "ACRouter extends the frontier upward in both ID and OOD with the highest AvgPerf and less cost than always choosing a . . . " | spend only | no | Table 3, Fig. 5, Fig. 6 |
| Jitkrittum et al. (2023) | other deferral rules, plus an oracle deferral rule as ceiling; not always-strong<br>*body:* "For each of these settings, we compute deferral curves that plot the classification accuracy versus the fraction of sa. . . " | calls | no | Fig. 2, Fig. 1 |
| Huang et al. (2026) | the single strongest backbone-role pair per dataset<br>*body:* "Best Agent selects the single strongest backbone–role pair per dataset, representing the best achievable performance . . . " | neither | no | Table 3 |
| Chen et al. (2023a) | best individual LLM (GPT-4)<br>*table caption:* "Cost savings by FrugalGPT to match the best individual LLM's performance." | spend only | no | Table 3, Fig. 5, Fig. 1c |
| Liu et al. (2026b) | fixed strong-model singletons<br>*body:* "The second regime is multi-model ensemble routing, where the router selects several complementary models and an aggreg. . . " | spend only | no | Tables 3–7, Fig. 3 |
| Taberner-Miller (2026) | fixed single-model baselines, per-prompt oracle as ceiling<br>*figure caption:* "(a) ParetoBandit with BudgetPacer (blue curve) traces a continuous quality–cost frontier by accepting a dollar budget. . . " | n/a, equal | no | Fig. 1, Table 2 |
| Zhou et al. (2026a) | the strongest cheap single agent, with cheap-oracle and full-oracle ceilings<br>*body:* "Because a deployable policy may run multiple diagnostic probes before returning a final ranked list, utility is define. . . " | spend only | no | Table 1, Table 2, Fig. 6 |
| Yang et al. (2026a) | always-strong as the cost-saving denominator; unrouted strong model in the dynamic track<br>*table caption:* "The unrouted Opus 4.6 baseline is separated below the routing policies, mirroring the cost-envelope row in Table 2." | n/a, equal | no | Table 2, Table 3 |
| Li et al. (2026)† | a same-backbone zero-shot router<br>*table caption:* "Main results on our enterprise developer-support dataset. †: BM25 + LLM-fewshot (Qwen3-8B). Cold: seed-only stores bef. . . " | neither | no | Tables 2–4 |
| Liu et al. (2025)† | full-context routing, stated as an upper bound<br>*body:* "It serves as an upper-bound baseline in terms of task success rate." | n/a, equal | no | Table 2, Table 10 |

Table 30: Baselines and attempt budgets in cited studies. *Spend only* matches dollar or token cost; *calls* includes a call-count axis; *neither* matches neither; *n/a, equal* denotes one call per request. † marks papers outside this comparison. Quotations are from each paper's body or a caption.

that bears on the identity's two terms (the first-slot cost $p_c - p_s$ and the rescue term) and on whether a benchmark-derived reward can carry either.

**Paying for a signal before acting.** Paying for a signal before committing compute is studied for expert mixtures (Saliencro et al., 2026) and for clarifying questions (Dong et al., 2026). Our judge cascade is the same shape (one model call buys information about whether to spend a second, more expensive one) and our negative result is about the *signal*, not the decision rule: given a perfect signal the verifier-gated effects of Section 3 are what a cascade earns, and the judge measures how much of that survives an observable substitute.

**Repairing the harness instead of switching the model.** A parallel literature asks which agent component to repair rather than which model to call (Xu et al., 2026; Chen et al., 2026a; Kang et al., 2026; Ling et al., 2026) and gates such edits statistically (Shawn, 2026; Moll et al., 2026; He & Yang, 2026). Two findings there mirror ours. Artifact effectiveness is backbone-dependent (Yu et al., 2026; Tang et al., 2026), the same shape as the routing effect's sign depending on which model sits in the second slot; and augmentation is often not worth its tokens at matched budget (Hajimiri et al., 2026), with evaluation environments now carrying cost records as a first-class view (Zheng et al., 2026); that is the budget-matching discipline our retry control applies one level down – the comparison has to hold attempts fixed, not just models. Judge bias is established (Lai et al., 2026), as is the difficulty of attributing evaluation shifts (Li, 2026), of timing interventions from observables (Modgil, 2026), and of single-culprit failure attribution (In et al., 2026); detecting drift at all is arguably ill-posed (Gower-Winter et al., 2026), and evaluation that is path-dependent rather than per-task raises the same attribution problem one level up (Yang et al., 2026b). Our judge-as-detector result is the special case where the intervention is an escalation and the observable is a transcript.

**Assigning models to roles, and selecting by search.** Two-slot routing is the smallest case of assigning models to roles within one task. Knowledge-aware co-reasoning across several specialised participants (Li et al., 2025b) is the multi-role version, and its evaluation faces the same confound ours does: adding a participant adds an attempt, so the control has to hold the number of attempts fixed rather than the number of distinct models. Separately, our honest-oracle measurement is a statement about selection: choosing the arm on the trials used to score it inflates the gain, and the inflation is not small. Search-based selection over a policy space with a dual-aware objective (Li et al., 2025a) faces the same hazard in a different setting: the selected policy's score on the selection data is an upper bound, not an estimate.

### A.25 The analysis pipeline and the checks it enforces

Every measured quantity in this paper is emitted by `make_tables.py` into a single macro file that the document `\inputs`. We describe the pipeline because several of the errors in Appendix C were pipeline errors rather than analysis errors, and the checks below exist to prevent recurrence of those errors.

**Two entry points, and which one the supplement can run.** `rebuild_all.sh` is the full-recomputation entry point: it starts from `runs/`, the raw per-trial rollouts, and runs the chain in dependency order (per-benchmark decomposition, the text-feature router, then the cross-benchmark axis and multiplicity control, then tables and macros, then figures, then the document), stopping on the first non-zero exit. `runs/` is about 385 MB and is not shipped, so inside the supplementary archive that script stops at its first step. What a reviewer can run there is the second entry point, table and figure regeneration from `results/`: `cd code && python3 make_tables.py && python3 figures.py`, then compile `paper/main.tex`. That path needs no rollouts, no model endpoint and no accelerator, and it still checks the paper's numbers rather than only rebuilding files, because `make_tables.py` emits every number into the macro file and fails if the document references a macro it did not define. Three quantities that can only be counted from the raw rollouts are cached in `results/` so that this second path is closed: `term_counts.json`, `nl_assertions.json` and `verdict_telecom_train74_s100.json`. The analysis scripts default `-out` to the empty string, meaning "print but do not write". Reproducible invocations must therefore pass an output path when refreshing result files.

**One dataset list, one primary BFCL pass.** `datasets.py` holds the canonical dataset list and names which BFCL pass is primary. Before it existed, the table generator and the figure generator each carried their own list, and they disagreed about which `telecom` run to use, so a figure and a table in the same paper reported different routing effects for the same dataset. The corpus-range macros assert that the file they read is the pinned pass.

**Required inputs, not optional ones.** Inputs the text depends on are declared required; a missing one aborts the build instead of silently leaving a macro empty. This was added after a whole tier of judge results went missing from a table without any error.

**Undefined-macro, uncited-entry and hand-typed-numeral checks.** After writing the macro file the generator scans the document for every \macro it references and refuses to emit if any is undefined. It also refuses if any bibliography entry is never cited, so the bibliography contains no unused entries. Then it runs *two* numeral passes, not one: bare numerals in running prose outside math mode, and hand-typed decimals *inside* inline mathematics, including values wrapped in `$...$`. Bare integers inside mathematics are still not scanned, and the AI-use statement says so. Three categories of literal are allowlisted: settings we chose, structural constants, and values produced by a guard probe rather than by a run. The generator prints every allowlist entry with its reason and whether that reason is checked programmatically. These checks do not scan bare integers inside mathematics.

**What the supplementary archive carries, and what it does not.** The archive includes three levels of supporting data. The result file behind every table, which lets Path A regenerate the paper's numbers. A flat per-rollout reward matrix, `results/rollouts.csv`, one row per logged rollout with its benchmark, dataset, task id, arm, trial, reward, the two cost fields and the termination reason: that file is what makes the *intervals* recomputable, because every interval, bootstrap $p$ and multiplicity survivor count here is a resampling of exactly those scalars, and Path A on its own does no resampling. And a cryptographic digest of every raw rollout *file*. The digests are per file, not per rollout, so they establish that a copy of `runs/` is byte-identical to ours and cannot check one rollout on its own; the reward matrix is what carries per-rollout content. The raw rollouts themselves are not shipped, which limits transcript-level reproduction. The matrix contains no `messages` field, no tool arguments and no model output, which is why it can ship when the transcripts cannot.

**Checks that assert a relationship, not just presence.** Three checks verify that two independently computed quantities agree, as follows: the identity's residual must stay below $10^{-12}$, four decades above the worst rounding the summation admits and ten orders of magnitude below the smallest effect we report; the bootstrap's replicate mean must match the point estimate to $0.005$ for difference statistics, whose bootstrap bias is zero in theory, and to $10\%$ of the accompanying interval width for the ratio statistics, whose bias is a genuine $O(1/n)$ effect and reaches $-0.021$ on the held-out split where the conditioning event has only 38 members. Both abort the build, and both are raised explicitly rather than through `assert`, so optimised interpreters cannot strip them. We verified each fires by perturbing the quantity it guards: reintroducing the degenerate-pair defect moves the bias to $+0.033$, and mis-specifying one term of the identity moves the residual to $2.4 \times 10^{-2}$; and the predicted cheaper cascade direction must equal the measured one on every dataset (Table 31). The temperature state of every arm is read from the handler registry and the generation logs rather than asserted, for the same reason.

### A.26 Additional accounting identities

**The self-rescue rate is the noise floor, rescaled.** $\mathbb{E}[s \mid s \text{ fails}]$ is not an independent construct. Over ordered pairs $(i, j)$, $i \neq j$, with $k_t$ the number of successful $s$-trials on task $t$,

$$\mathbb{E}[s \mid s \text{ fails}] = \frac{\sum_t k_t(n - k_t)}{(n-1)\, n\, T\, (1 - p_s)},$$

which collapses to $u_s/\big(n(1-p_s)\big)$ (and hence to $\mathrm{V}(s \to s) = p_s + u_s/n$ exactly, with $u_s$ the fraction of non-unanimous tasks) precisely when $2 \leq n \leq 3$, since $k_t(n-k_t)/(n-1) = 1$ for every $k_t \in \{1, \ldots, n-1\}$ only at $n = 2$ and $n = 3$. At $n = 4$ a task with $k_t = 2$ contributes $4/3$ and the simple form is wrong by up to a third. Every dataset here has $n = 3$, so we verified the

simple form to machine precision on all 4 $\tau^2$ datasets, all 5 pool models and BFCL, but that check cannot distinguish the two statements, which is why we state the general form. So Appendix B's per-arm noise floors and Section 4's mechanism are the same measurement in different units, which is why the self-rescue rate costs one extra rollout per task to report. It is not simply the noise floor, though: what enters the identity is recoverable variance *per failure*. The held-out `retail` split has the highest self-rescue rate (0.5263) and the *lowest* strong-arm noise floor (0.250), because its failure rate is small.

**Why we do not report a much larger routing effect that our data admits.** On `retail`, $\mathrm{V}(\texttt{sonnet45} \to \texttt{haiku45}) - \mathrm{V}(\texttt{haiku45} \to \texttt{haiku45}) = +0.1892$, much larger than the $+0.0315$ we do report. It is not a routing effect under our control, because its baseline retries the *weaker* model: $\mathrm{retry}(\texttt{haiku45}) = 0.6306$ against $\mathrm{retry}(\texttt{sonnet45}) = 0.7883$. We use the stronger model's retry as the quality baseline. Retrying the cheap model may still be relevant under a cost constraint, but it answers a different comparison.

We report the number anyway because it is it illustrates the effect of the baseline choice. The whole $+0.1892$ is explained by `haiku45`'s self-rescue rate of 0.0465: retrying a near-deterministic model recovers almost nothing, so switching is the only thing the second slot can do. The same arithmetic, with the same two models, gives $+0.0315$ when the second slot holds the model that *does* benefit from retrying. Nothing about the benchmark changed between those two numbers: only which model occupies the slot whose failures the second attempt has to recover.

**The reversed cascade matches the forward one by construction.** With binary reward and the symmetric ordered-pair set, $\mathrm{V}(k_1 \to k_2) = 1 - \Pr[k_1 \text{ fails} \wedge k_2 \text{ fails}]$, which is symmetric in $k_1, k_2$. Table 7 shows the two cascade directions agreeing on every dataset; that is an identity of the estimator, not evidence about routing. Order affects only cost, and the rule for which order is cheaper is worth stating carefully. Under verifier gating the escalation rate is exactly $1 - p_k$, so the cheap arm always escalates *more*, and "cheap-first is cheaper when the first slot escalates less" is never satisfied. Subtracting the two total costs, $C_c + (1 - p_c)C_s < C_s + (1 - p_s)C_c \iff C_c/C_s < p_c/p_s$: cheap-first is cheaper exactly when the price ratio is below the accuracy ratio. That holds on 3 of 4 datasets and fails on `telecom`, whose price ratio is only mildly favourable while its accuracy ratio is the corpus minimum (Table 31). Our generator now refuses to emit if the predicted direction disagrees with the measured one on any dataset.

This break-even condition compares the two cascade orders. The cost comparison in Section 8 instead uses strong-model retry as its baseline. The surviving claim is cheap-first against *retrying the strong model*, and subtracting those two total costs gives a different threshold: $C_c/C_s < 1 - p_s + p_c$. `telecom` violates the first and satisfies the second comfortably, which is exactly why it is the low end of the $1.18\times$–$2.08\times$ range the main text quotes rather than an exception to it. Table 31 reports both thresholds and the multiple by which the observed price ratio would have to rise to reach the second: $1.45\times$ on `telecom` to $3.63\times$ on `airline`. A vendor repricing of that size is not one we can rule out, but it is not a rounding error either, and the generator asserts the inequality on every dataset rather than leaving the range to be re-derived.

| | price | | accuracy | | | break-even | price | |
|---|---|---|---|---|---|---|---|---|
| dataset | $C_c$ | $C_s$ | $p_c$ | $p_s$ | $C_c/C_s$ | vs. retry | headroom | cheaper |
| retail train | 8.20 | 26.74 | 0.613 | 0.685 | 0.307 | 0.928 | 3.03 | cheap-first |
| telecom train | 25.56 | 60.48 | 0.219 | 0.607 | 0.423 | 0.612 | 1.45 | strong-first |
| airline train | 2.90 | 11.42 | 0.567 | 0.644 | 0.254 | 0.922 | 3.63 | cheap-first |
| retail test | 4.24 | 13.13 | 0.792 | 0.842 | 0.323 | 0.950 | 2.94 | cheap-first |

Table 31: Which cascade order costs less, and the two ratios that decide it. *cheaper direction* is measured from the logged per-policy costs; the prediction from $C_c/C_s < p_c/p_s$ agrees with it on every row, which `make_tables.py` asserts.

Thus the "routing effect" is a coverage difference between two draws from a model *pair* and two draws from one model; it contains no routing *decision*, which is deliberate: it is the verifier-gated upper bound.

## B MEASUREMENT PRACTICE

**Noise floors are per-arm.** By an arm's *noise floor* we mean its non-unanimous fraction: the share of tasks on which its three logged rollouts do not all agree, which is the smallest disagreement any two-slot statistic over that arm has to live with. The strong arm's is $5.7\times$ to $10.0\times$ the cheap one's, and unbounded on `airline`, where the cheap arm is unanimous on every task (Appendix A.13). A single pooled noise estimate is therefore wrong by close to an order of magnitude, and paired designs assuming one noise level mis-state their power. The direction is the opposite of the intuition that a stronger model is a steadier one, and by the identity of Appendix A.26 it is the same measurement as the self-rescue rate that drives Section 4.

**The per-task-maximum oracle is inflated only under a condition that almost never occurs.** It is often argued that taking $\max_k \bar{r}_{t,k}$ charges sampling noise to routing, and that the inflation grows as trials get fewer. That is true only when the arms trade wins across trials for some task. Across the four $\tau^2$ datasets that happens on 1 task in total, on `telecom` and nowhere else (Table 24); on BFCL it happens on 0 of 200. So $\mathbb{E}_{\text{trials}}[\max_k r] = \max_k \bar{r}$ holds exactly on three $\tau^2$ datasets and to within $1.4\%$ of tasks elsewhere, and the oracle is nearly independent of the number of trials. The out-of-sample penalty that *is* real is different in kind: selecting the arm on the same trials used to score it. On `telecom` that penalty turns a naive arm-selection gain of $+0.0183$ into an honest $+0.0046$.

## C NEGATIVE RESULTS AND FAILED CRITERIA

**Acceptance criteria and their timing.** We recorded five criteria and verified from version control that their thresholds never changed. Two passed: the step-versus-session admissibility ratio and the `retail` arm-selection oracle. Both were inherited from an earlier probe and were already marked as passed in the idea card's first commit. We therefore claim no pre-registration for them. The three criteria recorded before their experiments all failed.

The admissibility criterion used a threshold of $100\times$. The pooled ratio is $312\times$ and the maximum $992\times$, but `airline` reaches only $69\times$, leaving 1 domains below the recorded threshold. Replication of the arm-selection oracle failed on both additional domains: `airline`'s interval covers zero, and `telecom`'s corrected gain is $+0.0046$, compared with $+0.0183$ before correction, on 73 tasks. Its interval is $[-0.0274, +0.0457]$, with a point estimate below a tenth of the $0.05$ criterion. The prespecified held-out router scored $-0.0250$ and failed its criterion. Session-level ranking also failed when measured on individual logged datasets. For two underpowered evaluations, at 15 and 30 tasks, estimates fell within half a point of the thresholds while intervals covered zero. We added tasks without changing the criteria.

**Discarded runs and archived records.** An initial run used a step cap that truncated most rollouts, making the arm comparison sensitive to finishing speed. Those rollouts were not retained and their numerical results are not reported. Two later partial runs remain under `runs/_truncated/`: a `retail` run at another seed with two trials, and the 50-step `telecom` run analysed in Appendix A.5. Their generated count is Two. They are absent from `results/runs.jsonl`, which records scored runs. The ledger's `invalid` row is a snapshot of the temperature-pinned BFCL pool while it was still running, superseded by the completed run.

**Replication across benchmark families.** The three $\tau^2$ domains share a harness, grader design and user simulator. They therefore provide limited external replication for a claim about evaluation stochasticity. Self-rescue varies by $1.9\times$ across the four $\tau^2$ datasets and by $11.3\times$ when individual models are separated. Both extremes of the latter range occur within $\tau^2$; BFCL's $0.0989$ lies inside it. This within-family variability did not predict the cross-family sign difference, motivating the independent BFCL comparison.

**The negative `telecom` effect survives a higher step cap.** Restricting the 50-step pass to 14 tasks with normally terminated rollouts selects easier tasks and removes the observed difference. Raising the cap to 100 instead reduces truncated rollouts from 243 of 444 to 7. The routing effect becomes $-0.0913$ $[-0.1279, -0.0548]$, still significantly negative and larger in magnitude (Appendix A.5).

**Lexical routers and judge tiers.** All eleven router configurations have non-positive training gain with one label trial. With two label trials, the selected configuration becomes always-strong. Choosing the best of all eleven on the held-out split gives $+0.0250$, but this reuses the test set for selection; the prespecified result is $-0.0250$. Higher-tier judges have worse point estimates for escalation because they declare completion more often and miss failures. Their accuracy intervals overlap pairwise (Table 22), so no ordering by tier is established.

**Untested $\tau^2$ solo mode.** One hypothesis is that the language-model user simulator adds outcome variance absent from BFCL's scripted users. We patched $\tau^2$'s solo mode to make it importable but did not run it (Appendix A.1). Only `telecom` supports this mode; `retail` and `airline` reject it. Solo mode also changes the policy document, transfers user tools to the agent, adds a stop tool, disallows ordinary text output, and gives the agent its task ticket initially. A solo-mode run would therefore change several factors alongside the simulator. It could test whether the result persists under this bundle, but could not isolate the simulator's causal contribution. Its cost would be below a standard `telecom` run because it makes no simulator calls.

**The four-scalar sign predictor.** Profiling disjoint task samples gives macro sign accuracy $0.475$, below the constant always-retry baseline $0.623$; cross-dataset accuracy is $0.417$. Target self-consistency, $0.482$, does not upper-bound predictor accuracy. It measures $\Pr[\text{sign}(\theta_{h_1}) = \text{sign}(\theta_{h_2})]$, whereas prediction accuracy measures $\Pr[\text{sign}(\hat{\theta}(h_1)) = \text{sign}(\theta_{h_2})]$. A constant predictor can exceed self-consistency: always-retry reaches $0.623$, and the best constant sign predictor reaches $0.727$ macro-averaged. Choosing the majority sign achieves $\max(\Pr[\theta > 0], \Pr[\theta \leq 0]) \geq 1/2$. Self-consistency diagnoses sign instability, but it does not explain away the rule's poorer performance than a predictor that uses no profiling data.