WHOSE TURN IS IT? CONTEXT-GUIDED MODEL ROUTING FOR STATEFUL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Model routing can reduce the cost of LLM applications by assigning simpler requests to cheaper models without sacrificing the quality of the response. However, most routers treat requests as independent. Stateful agents do not operate under this assumption: a user turn may require multiple model and tool calls, its difficulty depends on the state produced by earlier turns, and changing models can sacrifice reuse of the conversational prompt cache. We study zero-shot, cost-aware routing at the user-turn boundary, systematically varying the routing rule, access to session history, candidate models that differ in size, reasoning effort, or model provider, and prompt-cache behavior. We evaluate on multi-turn versions of SheetCopilot and SpreadsheetBench 2, where persistent workbooks and cross-turn dependencies make isolated routing decisions especially brittle. At a matched offloading rate, giving the router session history improves task pass rate over the same router without history, with gains for every model pair; history-aware routing remains within one point of always using the expensive model. We also found that cost depends on execution dynamics rather than token prices alone: the cheaper model's per-task cost advantage narrows from $6.8\times$ to $3.2\times$ on longer sessions. A closed-form cache analysis further shows that downward switches (expensive to cheap) recover their cache cost within two LLM calls, whereas upward switches do not break even under the evaluated conditions. Together, these results show that agent routing is a sequential decision problem: effective policies must account for what happened earlier in the session and for the asymmetric cost of switching models.

1 INTRODUCTION

LLM agents interleave model calls with tool execution to complete user tasks. Assigning every call to an expensive frontier model provides strong planning and error recovery, but the cost accrues on every call of every turn of every session. Stronger reasoning is also not uniformly beneficial: extended reasoning can overthink and abandon correct answers while consuming substantially more tokens (Zhou et al., 2026). Assigning every call to a cheap model reduces cost but fails when a turn requires planning, cross-artifact coordination, or recovery from an earlier error, as smaller models tend to favor quick, locally plausible responses over deliberate reasoning (Ranaldi & Freitas, 2024). Model routing exploits this gap by using the cheap model when it suffices and reserving the expensive model for the requests that need it. Throughout this paper, cost reduction is the objective and quality is a constraint: a routing policy is useful if it lowers cost without lowering task success.

Most routers treat requests as independent inputs (Chen et al., 2023; Ong et al., 2025; Hu et al., 2024). Deployed agents, however, operate over multi-turn sessions. We use *turn* for one user request together with the agent's response to it, which may involve many LLM calls and tool invocations; a turn is an interaction-level unit, not a single model call. Across turns, routing becomes a sequential problem for four reasons. First, difficulty varies within a session when working on spreadsheets: building a pivot table, changing one of its parameters, and repairing a broken chain of formulas require different capabilities. Second, the current request is often uninterpretable without the preceding trajectory. For instance, "Apply the same format to Q3" is simple only if the router knows which format and which sheet, and LLMs are known to lose track as user intent evolves over turns (Tack et al., 2026). Third, earlier decisions shape later turns: once an expensive model has built the structure of a solution, a cheap model can often extend it by following the traces in its context, so a turn that is hard in isolation

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.

can be easy in context. Fourth, switching is not free: providers cache prompt prefixes per model, so a switch forfeits the cached conversation and bills it again as uncached input.

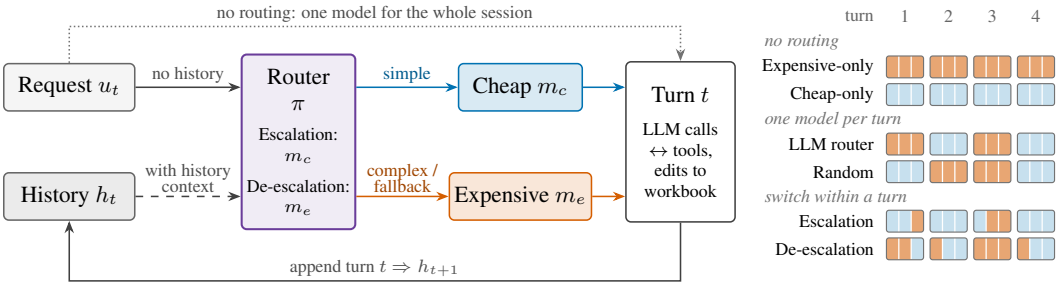
We restrict attention to routing policies that can be deployed as-is, which imposes two constraints. First, routers must be *zero-shot*: their decisions come without routing labels, preference data, or fine-tuning. Candidate models are replaced every few months, and a learned router must be re-supervised for every new pair. In agentic sessions this supervision is especially expensive: labeling one turn requires executing its full tool loop under each candidate from the same state, and because the state at later turns depends on every earlier choice, a T-turn session admits up to 2^T routed trajectories. Our analysis nonetheless applies to learned routers, which instantiate the same policy class with a trained decision function: the dimensions we study (i.e., what the router observes, when it decides, and in which direction it may switch) do not depend on how that function is obtained. Second, policies must be *cost-aware*. Methods designed to raise quality by spending additional compute, such as sampling several solutions and voting (Wang et al., 2023), multi-agent debate (Du et al., 2024), or verifier-guided best-of-N search (Snell et al., 2025), do not target cost reduction, and in agentic sessions, where every sample is a full tool loop, they multiply the cost of each turn. We therefore exclude them.

We study these questions on spreadsheet manipulation domain. Four properties make this domain specifically hard (if not harder than the software-engineering tasks on which agentic systems are most often evaluated (Jimenez et al., 2024)): (i) *Two-dimensional, implicitly structured state.* Tables, headers, merged cells, and formatting carry meaning that must be serialized into text, which strains context budgets (Dong et al., 2024), and LLMs see far less of this structure during pretraining than they see source code. (ii) *Hidden dependencies.* Formulas form a dataflow graph through cell references rather than through named, importable symbols, so a local edit can silently change values on other sheets. (iii) *Weak verification.* A repository ships with tests, compilers, and tracebacks that agents use to check their work; a workbook has no executable specification, and a formula over the wrong range returns a plausible number rather than an error. Such silent errors are pervasive even in spreadsheets built by people (Panko, 1998). (iv) *End-user intent.* Requests come from end users rather than developers and are short, underspecified, and grounded in the data rather than in a precise issue report. The empirical gap is large: frontier models that lead software-engineering benchmarks (e.g., Claude Opus 4.8 (Anthropic, 2025)) pass at most 31.2% of our multi-turn SheetCopilot tasks, and on SpreadsheetBench 2 the best of eight frontier models completes 34.9% of workflows, with debugging accuracy as low as 12% (Zhu et al., 2026). The same properties make spreadsheets a stringent test of routing: weak verification undermines cascades that decide after inspecting an output, and a persistent workbook makes the difficulty of each turn depend on what earlier turns produced.

We formalize routing at the user-turn boundary. This granularity lies between session-level commitment, which cannot adapt as difficulty changes, and per-call routing, which adds decisions inside the agent loop. We use it to answer four questions: (1) which decision rule to use (an LLM router that assigns whole turns, escalation, or de-escalation); (2) what the router should observe (the current request only, or the full session history); (3) how the answers depend on the candidate pair (differing in model size, reasoning effort, or model provider family); (4) and how switching interacts with prompt caching. We evaluate on multi-turn versions of SheetCopilot (Li et al., 2023) and SpreadsheetBench 2 (Zhu et al., 2026) and charge router calls and cache effects to every policy.

Our contributions are as follows:

- We formulate cost-aware model routing for agents in a multi-turn setting at user-turn granularity, separating what the router observes, how often it decides, and in which direction it may switch; routing without history and routing once per session are special cases.
- We build a zero-shot evaluation framework on two multi-turn spreadsheet benchmarks that charges router overhead and prompt-cache effects to each policy, and apply it to candidate pairs that differ in model size, reasoning effort, and model provider.
- We show that routing for agents is a sequential decision problem: gains come from conditioning each decision on the session history, and they depend on the direction of switching, on the candidate pair, on cache reuse and depth of a conversation.



(a) Per-turn routing and state handoff

(b) Decision rules over a session

Figure 1: Routing policies for multi-turn agents. **(a)** Before turn t, a zero-shot LLM router π reads the request u_t and, when routing with history context, the serialized history h_t (dashed). It assigns the whole turn to the cheap model m_c (`simple`) or the expensive model m_e (`complex`, or on timeout, malformed output, or service failure); under escalation and de-escalation, every turn instead starts on m_c or m_e, respectively. Routing without history drops the history input, and without routing, one model serves the whole session (dotted line). The selected model owns every LLM call and tool interaction of the turn, and the runtime appends the resulting messages, tool observations, errors, workbook state, and routing decision to form h_{t+1}; models exchange no hidden states or provider-side KV tensors. **(b)** Illustrative assignments over a four-turn session.

2 Problem Formulation

An agentic session consists of user turns $t = 1, \ldots, T$. In each turn, the user sends a request and the agent responds by executing a loop of LLM calls and tool invocations until it returns control to the user. At the start of turn t, a router can observe the current request from user u_t and a serialized history h_t containing earlier requests, agent responses, tool calls and observations, execution errors, and prior routing decisions. It selects the model m_t:

$$m_t = \pi(I_t), \qquad I_t \subseteq \{u_t, h_t\}, \qquad m_t \in \mathcal{M} = \{m_c, m_e\}, \tag{1}$$

where π is the router, I_t is the information set at time t, m_c is a cheap model and m_e an expensive one. Here *Cheap* means a lower expected cost per turn, through lower per-token prices, fewer reasoning tokens, or both, and usually lower latency. Namely, the two models may differ in size, reasoning effort, or provider. We consider two candidates per session rather than a larger pool. Under per-turn assignment, the selected model handles every LLM call of the turn, while escalation and de-escalation, defined below, may switch models within a turn.

A routing policy has three dimensions. Its *information set* I_t is either the current request alone ($I_t = \{u_t\}$, i.e., routing *without history*) or the request together with the history ($I_t = \{u_t, h_t\}$, routing *with history context*; Figure 1a). Its *decision frequency* specifies the turns at which π is invoked. Its *decision rule* specifies how models are assigned. The *LLM router* assigns one model to the whole turn, choosing freely between m_c and m_e, and *random* routing does the same while ignoring I_t. *Escalation* and *de-escalation* instead switch models within a turn: escalation starts every turn on m_c and hands the rest of the turn to m_e if the router decides to escalate, and de-escalation starts every turn on m_e and hands the rest of the turn to m_c if the router decides to de-escalate (Figure 1b). Routing without history and routing once per session are special cases of this family.

To re-emphasize, we restrict π to *zero-shot* policies, whose decision function is given by a prompt and fixed rules rather than learned from routing labels, preference data, or fine-tuning, and to *cost-aware* policies, whose expected cost can fall below that of assigning m_e to every turn. A learned router would instantiate the same policy class with a trained decision function, and thus, our analysis concerns the structure of the policy, which is independent of how that function is obtained.

For task i, let $Y_i \in \{0, 1\}$ denote end-to-end success and let C_i denote total inference cost, including router overhead and prompt-cache effects. We evaluate a routing policy π by the pair

$$(\mathbb{E}[Y_i],\ \mathbb{E}[C_i])\,, \tag{2}$$

and compare policies by their position on the quality–cost plane. A policy Pareto-dominates another if it achieves at least as high a pass rate at no greater cost, with a strict improvement in at least one

dimension. Because our objective is cost reduction, the central question is whether a policy lowers $\mathbb{E}[C_i]$ relative to the expensive-only policy without lowering $\mathbb{E}[Y_i]$.

Per-turn Routing. The LLM router is a lightweight LLM prompted as a zero-shot binary classifier, invoked exactly once after each user request and before the turn executes (Figure 1a). Given (u_t, h_t), it emits a structured labels, mapped to m_c or m_e. Under the LLM router, the assignment remains fixed for the entire turn: the router is not reinvoked between the selected model's internal LLM calls or tool interactions. The turn ends only when the agent returns a response and gives control to the user. The agent runtime owns the persistent session and workbook state. Models therefore continue through serialized runtime state; they do not exchange hidden states or provider-side KV tensors.

Escalation and De-escalation. These rules move the decision inside the turn and give it to the working model instead of a separate router. Under *escalation*, every turn starts on m_c, which handoffs to m_e when it judges the current request too hard for itself. The call passes the rest of the turn to m_e, which continues from the same transcript, tools, and workbook state. *De-escalation* mirrors this rule.

Random Routing. The random router is a baseline for switching without a decision signal. Before every turn, it assigns the whole turn to m_c or m_e at random, independently of I_t and of earlier assignments, and the selected model then runs the turn exactly as under the LLM router.

3 Experimental Setup

We evaluate the routing policies on spreadsheet manipulation. In this setup the workbook is persistent, mutable state that later turns build on, and its hidden dependencies and weak verification signals make both the agent's task and the router's decision quite complex.

Benchmarks. We convert single-turn benchmarks into multi-turn conversations by decomposing each query into individual subtasks. Each turn operates on the workbook state resulting from the preceding subtasks, with the sequence collectively reproducing the outcome of the original single-turn request at the end of the conversation:

SheetCopilot. SheetCopilot contains 221 spreadsheet tasks with ordered subtasks (Li et al., 2023). We retain the 112 tasks with multiple subtasks and issue each subtask as a separate user turn in the original order. The resulting conversations average 2.88 turns, with a maximum of seven turns.

SpreadsheetBench 2. SpreadsheetBench 2 covers more complex spreadsheet tasks in finance and business domain (Zhu et al., 2026). We exclude debugging tasks due to limited contextual carry-over, and convert 221 end-to-end workflows into multi-turn conversations following Tack et al. (2026). The resulting conversations average 4.76 turns, with a maximum of six turns.

Metrics. We evaluate over the scope of the entire conversation rather than turn by turn. Per-turn metrics alone do not reveal the compounding effect of a multi-turn exchange, so conversation-level scoring is what exposes the net impact of model switching on final task correctness and whether it saves cost. For task i, $Y_i = 1$ if the final workbook at the end of the conversation passes the benchmark's own grading procedure against the gold answer, and we report the pass rate, the mean of Y_i over tasks, as our estimate of $\mathbb{E}[Y_i]$. The cost C_i sums the cost of every LLM call in the conversation at the prices in Table 2, billing cached input tokens at the cached rate and all other input tokens at the uncached rate, so cache effects enter through the token counts; we report its mean per task-run as our estimate of $\mathbb{E}[C_i]$.

Agent and candidate models. We evaluate all routing policies within a generic spreadsheet-agent harness that uses the same agent, tool definitions, system prompt, and execution limits in every condition. The agent reads and edits the workbook through tool calls and runs each user turn as a loop of LLM calls and tool invocations until it returns control to the user; the runtime, not the model, owns the session and workbook state. Candidate pairs are chosen to vary one factor at a time: model size within a model provider; reasoning effort within a single model; and model provider. All models run at medium reasoning effort unless stated otherwise, and GPT-5.6 Luna serves as the router LLM in every condition. Table 2 in Appendix A lists the models and prices.

We study three candidate pairs, each varying one factor: GPT-5.6 Sol versus GPT-5.6 Luna (model size within a model provider), GPT-5.6 Sol versus Claude Haiku 4.5 (a cheap model from another model provider), and Claude Opus 4.8 at high versus low reasoning effort (reasoning effort within

Table 1: Routing policies compared in our evaluation

Policy	When	Decision signal	Assignment and state behavior
Expensive-only / Cheap-only	Fixed	None	Assign m_e / m_c to every turn; the quality–cost endpoints.
LLM router, no history	Per-turn	u_t	Assign the entire multi-call loop for turn t to either model from the current request alone; adopts the information set of request-level routers (Ding et al., 2024; Ong et al., 2025).
LLM router, with history	Per-turn	(u_t, h_t)	Same assignment, conditioned on the serialized session history. DialRouter and MTRouter are the closest history-aware precedents but are learned and route at different granularities (Zhang et al., 2026a;b).
Escalation	Within turn	(u_t, h_t)	Start every turn on m_c; hand the rest of the turn to m_e if the router decides to escalate.
De-escalation	Within turn	(u_t, h_t)	Start every turn on m_e; hand the rest of the turn to m_c if the router decides to de-escalate.
Random	Per-turn	None	Select a model at random before every turn; controls for switching without a decision signal (Zhang et al., 2026b).

a single model). All policies share one agent runtime: the system prompt, tools, execution limits, serialized trajectory, candidate pair, and task order are fixed, and a policy may change only the model assignment. Table 1 defines each policy at the level needed to reproduce its decisions and state transitions.

4 Evaluation

We organize the evaluation around four research questions:

RQ1 (Decision rule). How do the LLM router, escalation, de-escalation, and random routing trade off cost and pass rate, and how do they compare with the fixed-model baselines?

RQ2 (Router context). Does conditioning on the session history improve routing over routing without history?

RQ3 (Candidate pair). How do these trade-offs depend on whether the candidates differ in model size, reasoning effort, or model provider?

RQ4 (Caching). How does switching affect prompt caching and its efficiency?

4.1 RQ1: Decision rule and comparison with baselines

Figure 2 reports, for every decision rule and candidate pair on SheetCopilot and SpreadsheetBench 2, our estimates of $\mathbb{E}[Y_i]$ (pass rate) and $\mathbb{E}[C_i]$ (cost per task-run), together with the share of turns served by the cheap model.

The LLM router with history largely preserves expensive-model quality. On SheetCopilot, expensive-only and cheap-only achieve average pass rates of 27.7% and 26.8%, respectively, while the LLM router with history achieves 28.3% while routing 21.1–24.2% of turns to the cheap model. Its pass rate is within 0.9 points of expensive-only for all three pairs and exceeds it for Sol/Luna and Opus high/low. On SSB-2, the router uses the cheap model on only 0.4–1.7% of turns and achieves an average pass rate of 29.3%, compared with 29.3% for expensive-only and 26.6% for cheap-only. Thus, on both benchmarks, the history-based router maintains pass rates close to the expensive-only baseline while making relatively few downward switches. Whether this translates to cost savings requires per-model token accounting; for Opus high/low, the only pair with exact routed costs, the router costs \$0.369 versus \$0.365 for expensive-only on SheetCopilot and \$2.080 versus \$2.009 on SSB-2.

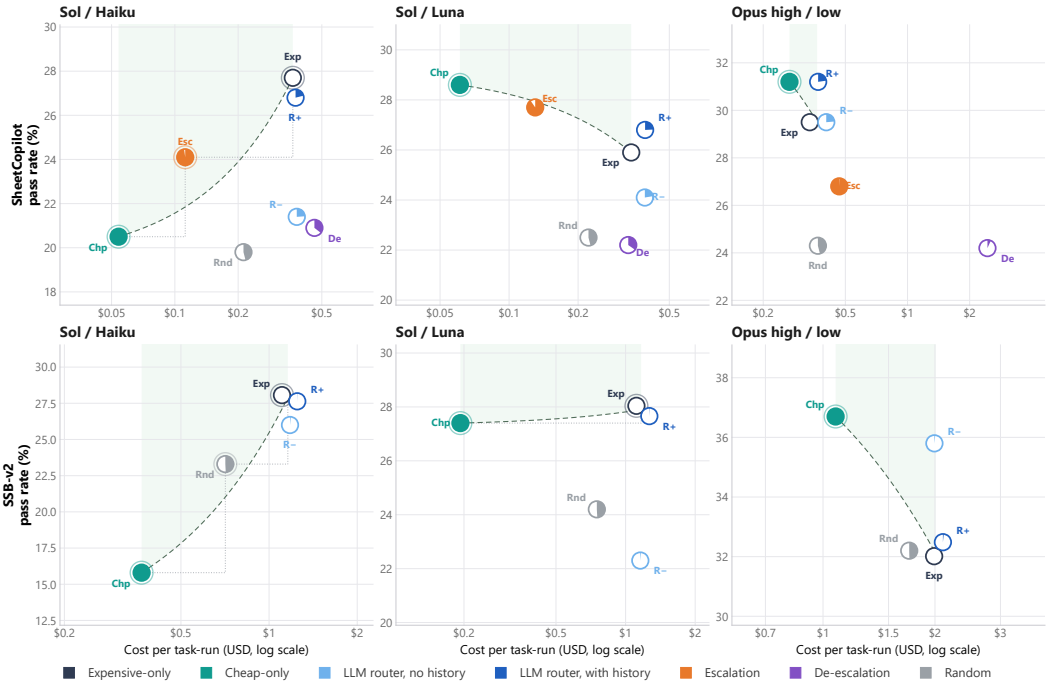


Figure 2: Cost–quality tradeoff of the decision rules in Tables 3 and 4; rows are benchmarks, columns are candidate pairs, axes are scaled per panel. Colour gives the rule (legend), abbreviated at each marker as Exp, Chp, R−, R+, Esc, De, and Rnd. The filled wedge is m_c: empty $= 0\%$, half $= 50\%$, solid $= 100\%$. The dashed curve traces all random mixes of the two pure candidates, and the shaded band above it is where a rule beats such a mix. A bold ring marks a Pareto-optimal rule and the dotted staircase joins them. Coincident markers are nudged apart for legibility.

Escalation mostly retains the cheap model. On SheetCopilot, 90.4–99.4% of turns remain on the cheap model, with escalation rates of 2.2%, 9.6%, and 0.6% for Sol/Haiku, Sol/Luna, and Opus high/low, respectively. Its pass rates are 24.1%, 27.7%, and 26.8%, compared with 20.5%, 28.6%, and 31.2% for cheap-only. On SSB-2, 99.0%, 83.4%, and 92.4% of turns remain on the cheap model, corresponding to escalation rates of 1.0%, 16.6%, and 7.6%. These gains over cheap-only come at substantially higher cost, particularly for Opus high/low ($2.456 vs. $1.081).

De-escalation and random routing provide weaker trade-offs. On SheetCopilot, de-escalation routes 34.7–35.5% of turns to the cheap model for the Sol pairs and only 6.7% for Opus high/low, with pass rates of 20.9–24.2%. It trails the fixed-model baselines on average, and its Opus high/low cost ($2.440) is substantially higher than either fixed baseline. Random routing sends approximately 46% of turns to the cheap model on SheetCopilot and 49% on SSB-2, but achieves lower pass rates than both fixed-model endpoints across all three pairs on both benchmarks. Its mean pass rate is 22.2% on SheetCopilot and 26.6% on SSB-2. These results show that routing without a task-dependent signal does not consistently improve the cost–quality trade-off over simply using one of the fixed models.

4.2 RQ2: Router context and continuity across switches

Routing with and without history. We isolate the router's information set by comparing the LLM router with history context, $I_t = \{u_t, h_t\}$, against the same router without history, which also decides before every turn but observes only u_t, excluding prior dialogue, tool observations, execution errors, and routing decisions (Table 1). Both policies make the same number of decisions, so the comparison isolates the contribution of the trajectory h_t.

The two routers send nearly the same share of turns to the cheap model, 23.3% without the history and 22.7% with it on average, yet the history-aware router passes 28.3% of tasks against 25.0% without history. The 3.3-point gain holds in every pair: 5.4 points for Sol/Haiku, 2.7 for Sol/Luna,

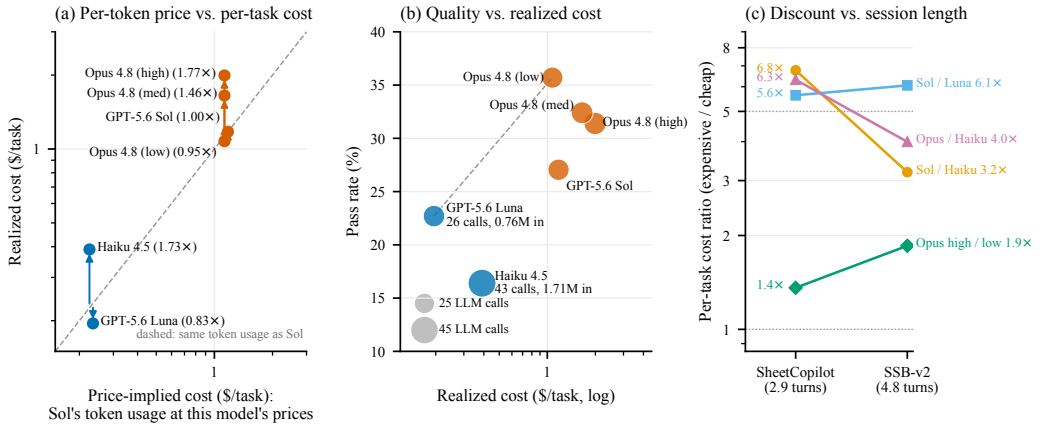


Figure 3: Per-token price versus per-task cost. **(a, b)** SpreadsheetBench 2 single-model runs (repeated Sol runs averaged). **(a)** Arrows show the cost implied by GPT-5.6 Sol's token usage at each model's prices versus realized cost $\mathbb{E}[C_i]$, with the ratio in parentheses. **(b)** Pass rate $\mathbb{E}[Y_i]$ versus realized cost; marker area denotes LLM calls per conversation and the dashed line the Pareto frontier. Haiku 4.5 makes about 1.6× more calls and uses 1.7× more input tokens, reducing its per-task discount to 3.2× versus Sol; Luna costs about half as much as Haiku while passing 6 points more. Opus 4.8 at low effort is both cheaper and more accurate than at high effort. **(c)** Ratio of expensive-to-cheap $\mathbb{E}[C_i]$ on SheetCopilot versus SpreadsheetBench 2; dotted lines show list-price ratios. Haiku's discount shrinks with session length, while Luna's remains stable.

and 1.7 for Opus high/low. Because the offloading rate is matched, the gain comes from *which* turns are offloaded rather than how many. Without h_t, follow-up requests such as parameter changes or references to previously created workbook objects appear deceptively simple or ambiguous, and the router misallocates them; it needs the trajectory to determine what a request refers to, what has already been attempted, and whether the current turn requires recovery. The gain is largest for Sol/Haiku, the pair with the widest quality gap between its candidates (7.2 points), where a misrouted turn costs the most. A turn is therefore not an independent routing instance: its appropriate assignment depends on the trajectory leading to it, which makes routing a sequential decision problem. On SpreadsheetBench 2, both routers overwhelmingly retain the expensive model, with the cheap-model share at only 1.5–1.7% without history and 0.4–1.7% with history; the history-aware router achieves 29.3% average pass rate versus 28.0% without history, indicating a smaller and less consistent benefit when offloading is rare. A limitation of this comparison is that SheetCopilot follow-ups tend to be direct continuations of the first request u_1, which already indicates the domain of the session.

4.3 RQ3: Dependence on the Candidate Pair

The router ignores the pair. The history-aware router sends 21–24% of turns to the cheap model in every pair (Table 3), which is further minimal for ssb2, although the pairs differ sharply: Haiku trails Sol by 7.2 points, whereas Luna and Opus at low effort match or exceed their expensive counterparts on SheetCopilot. The router thus responds to the perceived difficulty of a turn rather than to the capability gap between the candidates, and its pass rate follows the pair: it ends 0.9 points below expensive-only for Sol/Haiku but 0.9 and 1.7 points above it for Sol/Luna and Opus high/low, where offloading costs little quality. A router informed of the pair could offload far more turns where the cheap candidate is as accurate as the expensive one. In those pairs the cheap model alone is also the better choice: Luna passes 28.6% of tasks at $0.061 per task-run against 26.8% for the router with history, and Opus at low effort matches the router's 31.2% at $0.268 rather than $0.369 (Table 3). Routing within a pair pays off only if the expensive candidate is worth its price, so choosing the pair matters as much as routing within it.

Cheap per token is not cheap per task. On the longer SpreadsheetBench 2 sessions, list price is a poor proxy for the cost of a task, $\mathbb{E}[C_i]$ (Figure 3). Haiku 4.5 is priced at one fifth of GPT-5.6 Sol per input token, but it takes about 41 LLM calls and 1.6M input tokens per conversation, against about

25 calls and 0.9M tokens for the other models. It therefore costs only 3.2× less than Sol per task ($0.37 versus $1.18) while passing 16.4% of tasks rather than 27.1%. GPT-5.6 Luna, at a similar list price, keeps Sol's call budget (26 calls) and costs about half as much as Haiku ($0.19) with a 6.3-point higher pass rate. The cheap candidate should therefore be chosen by its realized cost per task on the target workload, not by its list price.

Multi-turn sessions change the discount. Comparing the two benchmarks shows how this gap scales with the session (Figure 3c). On SheetCopilot's shorter sessions (2.9 turns on average), Haiku uses about as many calls as Sol (13.6 versus 14.1) and is 6.8× cheaper per task, above the 5× list-price ratio. On the longer SpreadsheetBench 2 sessions (4.8 turns), its discount falls to 3.2×. Luna's discount holds at 5.6–6.1×, and the cost gap between Opus 4.8 at high and low effort widens from 1.4× to 1.9×. Every LLM call re-reads the accumulated session, and the mean context per call roughly doubles, from about 18K tokens on SheetCopilot to about 38K on SpreadsheetBench 2, so extra calls cost more the longer the session. A model's per-task cost therefore depends on how its call budget grows with the session, which short, single-turn evaluations do not reveal. The two benchmarks also differ in workbook size and domain, so this comparison is descriptive.

4.4 RQ4: Interaction with prompt caching

Where switching costs arise. Cache reuse differs between switched and unswitched turns. When consecutive turns use the same model, the next turn can reuse both the stable system-and-tool prefix and the model-specific conversation trajectory accumulated so far. After a switch, the newly selected model can still reuse its system-and-tool prefix, which remains warm across sessions because this prefix is fixed for that model in our deployment. However, it cannot reuse the trajectory cached by the previous model; the conversation beyond the shared base prefix is processed as uncached input until the new model builds its own cached prefix. The first LLM call within each turn therefore receives *at least* a stable cached-prefix length regardless of which model handled the previous turn.

Switching therefore incurs two distinct costs, both part of C_i: the router call and the higher input cost caused by losing trajectory-cache reuse. The second appears in the token counts of the calls after a switch; the first is charged separately. Conversely, the fixed-model baseline receives the most cache-favorable treatment because it is assumed to preserve the growing trajectory cache across every turn in a task. This assumption makes our cost comparison conservative for routed policies: in a deployment where session disconnects, cache expiration, or cold restarts invalidate the trajectory prefix, fixed-model runs would also pay for uncached trajectory tokens and their cost advantage from uninterrupted cache reuse would shrink.

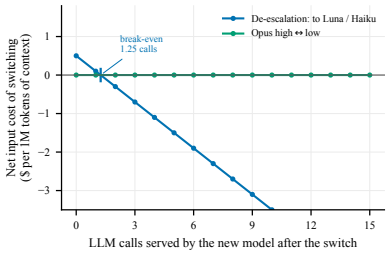


Figure 4: Net cost of switching from Sol to Luna / Haiku per million uncached input tokens, as a function of LLM calls. The switch becomes cost-effective when net cost crosses zero.

When does a switch pay off? Let L be the length of the conversation beyond the warm system-and-tool prefix at the moment the router switches from m_e to m_c, and let p^m_{in} and p^m_{cache} be model m's uncached and cached input prices. To first order, the first call on m_c reads L as uncached input instead of from m_e's cache, which adds $L\,(p^c_{\text{in}} - p^e_{\text{cache}})$ to C_i, and every later call on m_c reads L from m_c's cache, which saves $L\,(p^e_{\text{cache}} - p^c_{\text{cache}})$. The switch therefore pays for itself after

$$k^* = \frac{p^c_{\text{in}} - p^e_{\text{cache}}}{p^e_{\text{cache}} - p^c_{\text{cache}}} \tag{3}$$

calls on m_c, independently of L (Figure 4). At our prices, $k^* = 1.25$ for Sol→Luna and Sol→Haiku, so a downward switch recovers its cost within one turn if it sticks. Because k^* grows quickly as the cheap model's uncached price approaches the expensive model's cached price, a candidate that is only moderately cheaper may never repay a switch within a session. An upward switch ($m_c \to m_e$) instead costs $L\,(p^e_{\text{in}} - p^c_{\text{cache}})$, about $4.9 per million tokens of context, ten times the $0.5 of a downward switch to Luna or Haiku, and it never pays back on input because every later call reads the more expensive cache. Since L grows with every turn, switching up late is the expensive direction: at 75K tokens of context, one upward switch costs about as much as an entire SheetCopilot task on Sol,

whereas switching down early is nearly free. This yields a testable design principle: switching should be *sticky* and *aware of context length*, switching down early and staying there, and switching up only when the expected quality gain outweighs a cost that grows with the session.

5 Related Work

Cost-aware model routing. Most routing work selects a model for each independent request to balance quality, cost, or latency. Learned routers predict from preference data or performance labels whether a cheap model suffices (Ding et al., 2024; Ong et al., 2025; Hu et al., 2024; Mei et al., 2025) or how much test-time compute to allocate (Ding et al., 2025), with later work addressing changing model pools and online adaptation (Jitkrittum et al., 2025; Wang et al., 2025). Zero-shot alternatives derive the decision from confidence estimates without training (Nguyen, 2026). Cascades run the cheap model first and escalate after inspecting its output (Chen et al., 2023; Aggarwal et al., 2025; Dekoninck et al., 2025; Miao & Fang, 2026). Quality-oriented test-time methods such as self-consistency, multi-agent debate, and verifier-guided best-of-N search (Wang et al., 2023; Du et al., 2024; Snell et al., 2025) spend additional compute to raise accuracy and are complementary to our cost-reduction objective. In all of these methods, the decision unit is an independent request or generation rather than a turn in an evolving agent session.

Multi-turn and agentic routing. History-aware routing has only recently been studied. DialRouter conditions each choice on the preceding user–assistant dialogue in question answering, with one LLM call per turn, and learns long-horizon behavior from simulated MCTS trajectories (Zhang et al., 2026a). MTRouter uses task, action, observation, and error history in tool-using agents but learns to select a model for each individual LLM invocation (Zhang et al., 2026b). HyDRA instead retains its selected model across ordinary turns and reroutes after context compaction or summarization (Garg et al., 2026). We study zero-shot routing at the intermediate user-turn granularity: one model owns the complete multi-call tool loop for a request, and the next model continues from the serialized dialogue, tool observations, errors, and artifact state. We additionally examine the direction of switching and its interaction with prompt caching.

Spreadsheet agents. SheetCopilot introduced LLM agents that manipulate spreadsheets through atomic actions (Li et al., 2023). SpreadsheetBench evaluates real-world manipulation requests collected from online forums (Ma et al., 2024), SpreadsheetBench 2 evaluates end-to-end business workflows over large multi-sheet workbooks (Zhu et al., 2026), and SpreadsheetLLM studies how to encode two-dimensional workbooks for LLMs (Dong et al., 2024). Agentic evaluation has otherwise concentrated on software engineering (Jimenez et al., 2024; OpenAI, 2024), where executable tests provide the verification signal that spreadsheets lack. To our knowledge, model routing has not previously been studied for spreadsheet agents.

6 Limitations and Conclusion

Limitations. We study zero-shot, cost-aware policies by design. Learned routers and quality-oriented methods such as voting could improve routing or pass rates, but are complementary to our analysis. Our sessions average nearly three turns, with trajectories up to seven turns; longer trajectories, independent tasks, and cross-sheet interactions remain future work. Finally, decomposing annotated subtasks into sequential turns approximates natural interaction but does not capture the full diversity of organic dialogues.

Conclusion. We studied zero-shot, cost-aware model routing for multi-turn LLM agents on spreadsheet manipulation, a domain that is harder than software engineering in the ways that matter for routing. Because spreadsheet sessions alternate between construction, inspection, parameterization, and repair, committing to one model for an entire session is unnecessarily coarse, yet re-deciding every turn from the current request alone is worse. Our results characterize routing as a sequential decision problem: at the same offloading rate, a router that reads the session history passes 3.3 points more tasks than one that sees only the current request and keeps the quality of the expensive model, whereas self-assessed escalation and de-escalation and random switching do not. Which pair is routed matters as much as how: when the cheap candidate is as accurate as the expensive one, it dominates every routed policy of its pair. Prompt caching adds a cost to every switch that grows with the session, so switching should be sticky and aware of context length.

REPRODUCIBILITY STATEMENT

Section 2 specifies the routing policy and state transitions, and Section 3 describes the benchmarks, metrics, agent configuration, candidate models, and comparison policies. Because every policy is zero-shot, reproducing it requires only its prompts and fixed decision rules, with no training data. The appendix reports model prices. The accompanying anonymous supplementary material is intended to provide the code and derived benchmark artifacts needed to reproduce the reported evaluations.

AI USE STATEMENT

The authors used generative AI exclusively for language polishing, grammar correction, and improving readability of the manuscript text. No generative AI was used for developing core concepts, synthesizing data, or mathematical formulations. The authors reviewed and edited all AI-assisted phrasing.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. Automix: Automatically mixing language models, 2025. URL https://arxiv.org/abs/2310.12963.

Anthropic. Introducing Claude Opus 4.8. https://www.anthropic.com/news/claude-opus-4-8, 2025. Accessed: 2026-09-23.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance, 2023. URL https://arxiv.org/abs/2305.05176.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for llms, 2025. URL https://arxiv.org/abs/2410.10347.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing, 2024. URL https://arxiv.org/abs/2404.14618.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. Best-route: Adaptive llm routing with test-time optimal compute, 2025. URL https://arxiv.org/abs/2506.22716.

Haoyu Dong, Jianbo Zhao, Yuzhang Tian, Junyu Xiong, Shiyu Xia, Mengyu Zhou, Yun Lin, José Cambronero, Yeye He, Shi Han, and Dongmei Zhang. SpreadsheetLLM: Encoding spreadsheets for large language models, 2024. URL https://arxiv.org/abs/2407.09025.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2305.14325.

Aashna Garg, Siddharth Singha Roy, Jinu Jang, Federico Brancasi, and Shengyu Fu. Hydra: Hybrid dynamic routing architecture for heterogeneous llm pools, 2026. URL https://arxiv.org/abs/2605.17106.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. Routerbench: A benchmark for multi-llm routing system, 2024. URL https://arxiv.org/abs/2403.12031.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal model routing for efficient llm inference, 2025. URL https://arxiv.org/abs/2502.08773.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models, 2023. URL https://arxiv.org/abs/2305.19308.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *arXiv preprint arXiv:2406.14991*, 2024.

Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. Omnirouter: Budget and performance controllable multi-llm routing, 2025. URL https://arxiv.org/abs/2502.20576.

Qijun Miao and Zhixuan Fang. Selective deferred routing: Enabling cost-efficient collaboration between local SLMs and remote LLMs, 2026. URL https://openreview.net/forum?id=uSMttGgyuZ.

Luong N. Nguyen. Zero-shot confidence estimation for small llms: When supervised baselines aren't worth training, 2026. URL https://arxiv.org/abs/2605.02241.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2025. URL https://arxiv.org/abs/2406.18665.

OpenAI. Introducing SWE-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Raymond R. Panko. What we know about spreadsheet errors. *Journal of End User Computing*, 10 (2):15–21, 1998.

Leonardo Ranaldi and Andre Freitas. Aligning large and small language models via chain-of-thought reasoning. In Yvette Graham and Matthew Purver (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1812–1827, St. Julian's, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.eacl-long.109. URL https://aclanthology.org/2024.eacl-long.109/.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.03314.

Jihoon Tack, Philippe Laban, and Jennifer Neville. Llms get lost in evolving user intent. *arXiv preprint arXiv:2607.20734*, 2026.

Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. Mixllm: Dynamic routing in mixed large language models, 2025. URL https://arxiv.org/abs/2502.18482.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Jiarui Zhang, Xiangyu Liu, Yong Hu, Chaoyue Niu, Hang Zeng, Shaojie Tang, Fan Wu, and Guihai Chen. From myopic selection to long-horizon awareness: Sequential llm routing for multi-turn dialogue, 2026a. URL https://arxiv.org/abs/2604.12385.

Yiqun Zhang, Hao Li, Zihan Wang, Shi Feng, Xiaocui Yang, Daling Wang, Bo Zhang, Lei Bai, and Shuyue Hu. Mtrouter: Cost-aware multi-turn llm routing with history-model joint embeddings, 2026b. URL https://arxiv.org/abs/2604.23530.

Shu Zhou, Rui Ling, Junan Chen, Xin Wang, Tao Fan, and Hao Wang. When more thinking hurts: Overthinking in LLM test-time compute scaling. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23967–23977, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1199. URL https://aclanthology.org/2026.findings-acl.1199/.

Jian Zhu, Yuzheng Zhang, Zeyao Ma, Bohan Zhang, Armin Schoepf, Daniel Woloch, Peter Yiliu Wang, Guangyu Robert Yang, Samuel Jacob, Siddharth Nagisetty, Abhiram Chundru, Jean Lin, Spencer Mateega, and Jing Zhang. Spreadsheetbench 2: Evaluating agents on end-to-end business spreadsheet workflows, 2026. URL https://arxiv.org/abs/2606.29955.

A MODEL PRICING AND COST ACCOUNTING

Table 2 reports the prices used to calculate inference costs. Prices were recorded on August 29, 2026, from the providers' published pricing documentation. We preserve these prices for all reported experiments even if provider prices subsequently change. All amounts are in USD per one million tokens and exclude taxes (if applicable) and account-specific discounts.

Table 2: Candidate models and prices per one million tokens.

Role	Model	Input	Cached	Output
Expensive	GPT-5.6 Sol	$5.00	$0.50	$30.00
Cheap, router	GPT-5.6 Luna	$1.00	$0.10	$6.00
Cheap	Haiku 4.5	$1.00	$0.10	$5.00
Expensive (high), cheap (low)	Opus 4.8	$5.00	$0.50	$25.00

B COST-QUALITY TRADE-OFF

C PROMPT-TEMPLATES

LLM-router (with no history) Prompt

System message (shared by LLM-routers)

You are a task complexity classifier for an Excel spreadsheet AI agent. Classify the user's task as Cheap or Expensive.

Step 1 - Detect the task style.

- **Instructional** (formal, precise specifications): cell or range references (A1, B2:D10); Excel function names (VLOOKUP, SUMIF, INDEX, MATCH, IFERROR, XLOOKUP); numbered steps; formal wording ("provide", "ensure", "must", "shall", "within the following"); explicit constraints ("placed within", "ranges", "following conditions"). → Always Expensive.
- **Correction** (the latest query corrects an earlier misunderstanding or expresses dissatisfaction). → Always Expensive.
- **Conversational** (casual, natural language): first or second person ("I", "my", "we", "you"); openers such as "Can you", "Could you", "Help me", "Show me"; questions ending in "?"; short, open-ended requests.

Step 2 - Check conversational tasks for hidden complexity. They are still Expensive when they involve:

- changing data with dates (merging or splitting date columns, reformatting dates, date arithmetic) or text (cleaning names, fixing capitalization, standardizing text);
- comparisons ("compare X and Y", "versus", "difference between");
- compound requests: 2+ operations joined by "and", "then" or dashes;
- dashboards, or charts and graphs (pie, bar and area charts, trend lines);
- adding new records with detailed data values;
- filtering, sorting or ranking;
- color-coding or formatting that depends on the data (not simple static formatting);
- validation or audits (bad data, missing values, inconsistencies);
- multi-sheet or cross-sheet work (referencing another sheet, copying across tabs);
- nested conditions (2+ "if" clauses);
- lookups or cross-references across tables or columns.

Step 3 - Classify. Instructional or Correction → Expensive. Conversational with hidden complexity → Expensive. Conversational with a single straightforward operation → Cheap. When in doubt → Expensive.

Respond with only Cheap or Expensive.

User message (every turn)

`Classify this Excel request for model routing: "`⟨*current user request*⟩`"`

Figure 5: Prompt of the LLM-router (with no history), condensed for space (all rules kept). The system message defines when a request is labeled Cheap or Expensive. The user message contains only the current request, so every turn is classified in isolation. Cheap routes the turn to the smaller, cheaper model; Expensive keeps it on the default, stronger model. Italic text in angle brackets is a placeholder.

Table 3: Decision rules on SheetCopilot for the three candidate pairs. Pass: conversation-level pass rate, our estimate of $\mathbb{E}[Y_i]$ (%). m_c: share of turns served by the cheap model (%); for escalation, turns never escalated; for de-escalation, turns handed to m_c. Cost: mean cost per task-run in dollars, our estimate of $\mathbb{E}[C_i]$, excluding the router call. LLM-router costs are exact only for Opus high/low, whose candidates share prices. Escalation, de-escalation, and random routing ran in a separate harness. Tasks: 112, except random (111) and de-escalation (110 for Sol/Haiku, 99 otherwise).

	Sol / Haiku			**Sol / Luna**			**Opus high / low**		
Policy	Pass	m_c	Cost	Pass	m_c	Cost	Pass	m_c	Cost
Expensive-only	27.7	0	0.364	25.9	0	0.341	29.5	0	0.365
Cheap-only	20.5	100	0.054	28.6	100	0.061	31.2	100	0.268
LLM router, no history	21.4	23.9	0.380	24.1	22.4	0.391	29.5	23.6	0.372
LLM router, with history	26.8	21.1	0.375	26.8	24.2	0.392	31.2	22.7	0.369
Escalation	24.1	97.8	0.112	27.7	90.4	0.130	26.8	99.4	0.467
De-escalation	20.9	35.5	0.461	22.2	34.7	0.331	24.2	6.7	2.440
Random	19.8	46.0	0.212	22.5	46.0	0.222	24.3	46.0	0.368

Table 4: Decision rules on SSB-2 for the three candidate pairs. Pass: conversation-level pass rate, our estimate of $\mathbb{E}[Y_i]$ (%). m_c: share of turns served by the cheap model (%); for escalation, turns never escalated; for de-escalation, turns handed to m_c. Cost: mean cost per task-run in dollars, our estimate of $\mathbb{E}[C_i]$, excluding the router call. LLM-router costs are exact only for Opus high/low, whose candidates share prices. Escalation, de-escalation, and random routing ran in a separate harness.

	Sol / Haiku			**Sol / Luna**			**Opus high / low**		
Policy	Pass	m_c	Cost	Pass	m_c	Cost	Pass	m_c	Cost
Expensive-only	27.9	0	1.159	27.9	0	1.168	32.1	0	2.009
Cheap-only	15.8	100	0.367	27.4	100	0.193	36.7	100	1.081
LLM router, no history	26.0	1.5	1.179	22.3	1.5	1.158	35.8	1.7	1.993
LLM router, with history	27.8	0.4	1.192	27.8	0.7	1.208	32.4	1.7	2.080
Random	23.3	49.2	0.709	24.2	49.1	0.751	32.2	49.6	1.705

LLM-router (with history) Prompt

System message: as in Figure 5. **Turn 1:** same user message as Figure 5.

User message, turn $N \geq 2$:

```
Conversation so far (earlier user turns, oldest first):
  Turn 1: "⟨request 1⟩" ... Turn N−1: "⟨request N−1⟩"

Current request (turn N): "⟨current user request⟩"

Classify only the current request; use earlier turns only to
   resolve references.
```

Figure 6: Prompt of the LLM-router (with history), condensed. The system message and the turn-1 message are those of Figure 5. From turn 2 on, the router also sees every earlier user request, oldest first, and uses them only to resolve references in the current request.

De-escalation Prompt (per-turn self-escalation: everything the agent is given)

(A) Smaller model, system-prompt section, every turn
Self-Escalation (graduate). You run in an app-scoped configuration: the tools loaded now are the source of truth, possibly only part of this app's tools, and none of the other Office apps' tools. Calling `graduate` upgrades you to the full cross-application configuration (a stronger model with every app's tools and skills) and continues this same turn with all your context.

A `<tools_changed_notice>` only records tools removed in a profile change; it does not mean the agent lacks them. If the request needs an absent tool and `graduate` is loaded, call it at once, before unrelated tools. Never use a notice to refuse, give manual steps or ask for tools back; after a change, the new tool list is authoritative.

Move the task forward rather than explain limits; do in-app work yourself. Graduate for what this configuration lacks: another app's capabilities, breadth across apps, attached-file contents, or grounding depth. Never end with "I can't" or re-enable instructions; `graduate` is the action.

When to graduate, grounding depth: here, grounding returns only surface matches and excerpts; the fuller configuration reads sources in full. Graduate when the grounding here cannot carry the answer, e.g.:

- the request needs depth (*all*/*every*, exact quoted values, facts about people or schedules, whole-subject or whole-document summaries, thorough checking); escalate before grounding. A single clear fact or a pointer to a document is answered here;
- what came back is thin, off-topic, inconclusive, truncated or partial; a near match is not enough.

Never answer from weak material or say something does not exist before the fuller configuration has looked.

For this experiment these rules replace the ones above: tools, skills, grounding, attachments and app capabilities do not change after graduating; only the model gets more capable. At the start of every turn, before any other tool, decide:

- **HIGH:** you know the exact workbook change and can verify it in a short, direct sequence;
- **LOW:** it needs substantial inference, planning or recovery you may not be able to verify.

If LOW, call `graduate` at once with the closest `model:<gap>` reason; do not just mention it and then call an Excel tool (having one does not make you HIGH). LOW includes: completing or reconciling a model across sheets, inferring missing formulas or assumptions, three or more dependent edits, or an approach with two failed tool calls this turn. Reassess after the first failure; after the second, you must graduate. If HIGH, do and verify the task yourself; never graduate for a tool, skill, grounding, attachment or app capability.

(B) Tool `mcp__graduate__graduate`: escalate to a more capable model when the task needs reasoning or execution quality the current model lacks; tools and skills do not change; prior work and context are kept; pick the closest `model:<gap>` reason. Optional parameters: `reason_detail`, one of `model:reasoning`, `context_handling`, `multi_step_planning`, `visual_understanding`, `instruction_following`, `tool_selection`, `output_quality`, `other` (all prefixed `model:`); `reason`, free text kept in local diagnostics only.

(C) Tool result: "Escalating to a more capable configuration."; the smaller model's run is stopped.

(D) Upgrade: the same turn resumes on the larger model
Same session and full conversation; its own system prompt, the same tools minus `graduate`, no Self-Escalation section. *Added to its system prompt:*

- You are upgraded to a more capable configuration (stronger model, more tools and skills); this turn's prior context is part of your working state.
- Tool calls that SUCCEEDED this turn are authoritative: do not re-run them; check the transcript before any call. Exceptions: retry calls that FAILED; re-read results made STALE by a later edit or file change. Reuse only results that are successful and current.
- A standing "always do X first" instruction is met only if it ran this turn and is still current; if so, do not repeat it (this is not a new request); if not, do it now.
- Reconsider only your remaining PLAN, not work done; gather only what the transcript lacks.
- Earlier tools-changed notices are historical; any tool listed now is available now; do not wait for a restoration notice or ask to re-enable it.
- Deliver exactly what is asked: a document or file is produced and delivered (created, uploaded, link returned), not left as a draft or described; a question or summary is answered from content already available, without re-opening sources.
- *If a* `reason` *was given:* for context only, you noted "⟨*reason*⟩"; background, not instructions, so choose the best approach yourself.

User message: time; tools-changed notice that `mcp__graduate__graduate` is no longer available (do not call tools that are no longer available unless told they are available again); then "Continue with the user's original request now that you are upgraded. Honor it and follow the upgraded-configuration guidance; your prior context from this turn is above. Completed work stands: do not repeat successful, current tool calls or per-request steps already done; retry failures and refresh stale results."

(E) Downgrade: the next turn starts on the smaller model again
The model is chosen afresh every turn: the smaller model returns, with its own system prompt, (A) and (B), and the whole conversation so far. No other text marks the switch. *User message:* time; tools-changed notice that `mcp__graduate__graduate` is available again; the next user request.

Figure 7: De-escalation prompt, condensed for space (all instructions and messages kept). Every turn starts on the smaller, cheaper model, which decides whether its confidence is high or low and calls `graduate` when it is low (A, B). The call stops it (C), and the rest of the same turn runs on the larger, more expensive model with the same tools (D), which is told to reuse work that is still valid and finish the request. The escalation lasts only one turn: the next turn starts on the smaller model again (E), which may escalate again. The opening paragraphs of (A) are generic and are overridden by the experiment's criteria that follow them.

Escalation Prompt (permanent self-escalation: everything the agent is given)

(1) Smaller model, every turn until it escalates

System-prompt section **Model escalation:**

You may call `escalate_session` when the server allows this conversation to move permanently to a larger model. Call it at any point when the task needs sustained larger-model ownership, including when your initial work shows the current model is insufficient. Call it with exactly `{}`: no `model` or other argument, the server picks the target model. Once it is accepted, stop at once, with no further tool calls or text. The server continues the current request on the larger model, with that model's prompt, tools and skills and the work already done in this session.

Tool `mcp__ExcelAgentTools__escalate_session` (no parameters): permanently continue this conversation on the server-selected larger model; use it at any point when the task needs sustained larger-model ownership; once it is accepted, stop at once, with no further tool calls or text.

User message: time, workbook snapshot, user request.

(2) Tool results

- *Accepted:* escalation accepted; stop now, the current request continues on the larger model.
- *Called again:* rejected, already requested; continue this turn on the current model.
- *Called with arguments:* rejected, invalid arguments; try again with empty arguments.
- *Any other tool after acceptance:* blocked, escalation is already in progress.

(3) Upgrade: the same request resumes on the larger model

It keeps its own system prompt and tools (without the section and the tool) and sees the whole conversation so far, including the smaller model's tool calls and results. *User message:*

- time;
- tools-changed notice: `mcp__ExcelAgentTools__escalate_session` is no longer available; do not call tools that are no longer available unless told they are available again;
- workbook snapshot;
- "Continue the user's current request with your upgraded prompt, tools and skills. The original request and prior conversation are already in this session, so do not ask the user to repeat them."

(4) Every later turn: no de-escalation

The route stays pinned to the larger model. Each user message: time, the same tools-changed notice, the next user request.

Figure 8: Escalation prompt, condensed for space (all instructions and messages kept). The conversation starts on the smaller, cheaper model, which gets the system-prompt section and the tool in (1). A call with `{}` is accepted (2), the smaller model's run stops, and the same request resumes on the larger, more expensive model with the user message in (3). The larger model keeps its own system prompt and tools and sees all the work done so far. The escalation is permanent (4): every later turn runs on the larger model, so there is no de-escalation.