AGENTBATON: EFFICIENT MODEL COLLABORATION IN AGENTIC TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Solving long-horizon agentic tasks requires language agents to interact with their environments over multiple steps, and reliable performance often comes from the expensive **capable models** available through cloud providers. Meanwhile, alternative **efficient models** with inferior performance but a much lower cost are emerging, including lower-tier cloud models and smaller open-weights models deployable on local hardware. This raises a natural question that remains less explored than traditional query-level routing: can capable and efficient models collaborate within a single long-horizon agentic task to substantially reduce cost while preserving task accuracy? In this paper, we systematically investigate key design choices for such collaboration through model handoffs, where one model takes over from another, focusing on handoff direction and context transfer. We find that efficient-to-capable handoffs generally underperform capable-only execution with limited or even no cost reduction, whereas capable-to-efficient handoffs at suitable stages of task progress largely preserve or even improve accuracy while substantially reducing cost. These findings suggest that a one-time, simple, well-timed handoff strategy is good enough for cost-effective agent collaboration, motivating **AGENTBATON**, a lightweight detector that learns when to hand off from recurring patterns in task progress. Across 11 model pairs and two representative benchmarks, AGENTBATON reduces API costs by 65.9% on average while largely preserving task accuracy, with accuracy gains of up to 8.8 percentage points over capable-only execution. Experiments across various reasoning-effort settings further show that AGENTBATON extends the empirical accuracy–cost Pareto frontier.

1 INTRODUCTION

Language-model agents tackle long-horizon tasks through repeated interactions with their environments, using tools, observing feedback, and revising their actions over multiple steps (Yao et al., 2022; Yang et al., 2024a). Reliable and state-of-the-art performance is often achieved by expensive **capable models** available through cloud providers (Anthropic, 2026b; OpenAI, 2026a), whose repeated invocation can incur substantial inference costs. Meanwhile, alternative **efficient models**, including lower-tier cloud models (OpenAI, 2026b) and smaller open-weight models (Qwen Team, 2026c) deployable on local hardware such as NVIDIA DGX Spark and Apple Mac Studio, offer much lower costs but often achieve lower task accuracy when used alone.

Prior work on combining heterogeneous models has largely focused on query-level routing and cascading for individual requests (Chen et al., 2023; Ong et al., 2025; Chen et al., 2024; Hu et al., 2024). However, assigning a task to a model solely based on its initial query falls short in long-horizon agentic tasks. In these environments, actions alter the workspace and yield dynamic feedback, meaning the actual capabilities required evolve across turns rather than being determined upfront. This necessitates collaboration throughout the task trajectory. For example, a capable model might draft an initial plan or produce key intermediate results for an efficient model to build upon. Conversely, an efficient model could handle preliminary steps like information gathering before handing off to a capable model for more demanding steps. Both directions are intuitively plausible, leaving open the question of which paradigm best balances accuracy and cost.

Beyond the handoff direction, transferring interaction history introduces another trade-off. Directly including the full history in context ensures the receiving model understand prior progress and

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.

reuse existing ideas, but increases the prefill sequence length at handoff and, while that history is retained, the input sequence length in subsequent turns. Moreover, exposing the receiving model to its predecessor's reasoning and actions may bias its subsequent decisions, potentially carrying forward unproductive assumptions or strategies.

To understand how handoff direction and context transfer affect the accuracy–cost trade-off, we evaluate both handoff directions across multiple capable–efficient model pairs on two agentic benchmarks, Terminal Bench 2.1 (Merrill et al., 2026) and DeepSWE v1.1 (Huang et al., 2026). For each direction, we compare two context transfer methods: directly including prior interaction history in the receiving model's context, or providing structured execution history through a log file that the receiving model can inspect as needed using standard commands such as `cat`, `grep`, and `sed`. In both cases, the receiving model continues from the workspace produced by its predecessor.

The experiments reveal an asymmetry between the two handoff directions. Efficient-to-capable handoffs generally underperform capable-only execution, sometimes even at higher cost, despite benefiting from logfile transfer over in-context transfer. In contrast, capable-to-efficient handoffs at suitable stages of task progress can largely preserve capable-only accuracy while substantially reducing cost, although the preferred context-transfer method varies across models. These findings suggest that the progress established by a capable model can enable an efficient model to complete tasks that it struggles to solve independently. This one-way dependency also implies that per-turn routing may be impractical, as frequent switching risks accumulating trajectory errors that the capable model must repeatedly expend effort to recover from.

Our case studies help characterize these handoff opportunities: the capable model has often produced a substantive implementation or artifact and obtained task-relevant feedback, leaving primarily localized repair, validation, or finalization. Motivated by these recurring patterns, we introduce **AGENTBATON**, a lightweight detector trained to identify suitable handoff points from execution history. The capable model begins the task, and AGENTBATON evaluates progress after each turn; once it predicts a suitable handoff point, the efficient model takes over and continues until termination. Across 11 model pairs and two benchmarks, AGENTBATON reduces API costs by 65.9% on average while largely preserving task accuracy, with gains of up to 8.8 percentage points over capable-only execution. Experiments across reasoning-effort settings further show that AGENTBATON extends the empirical accuracy–cost Pareto frontier.

To sum up, we make three main contributions: **1)** We systematically analyze how handoff direction, context transfer, and task progress affect the accuracy–cost trade-offs of model collaboration within long-horizon agentic tasks. **2)** We introduce AGENTBATON, a lightweight learned detector that identifies suitable capable-to-efficient handoff points online based on recurring patterns in agent execution. **3)** We evaluate AGENTBATON across multiple model pairs and two agentic benchmarks, demonstrating substantial API cost reductions with largely preserved accuracy and improved accuracy–cost trade-offs across reasoning-effort settings.

2 EXPERIMENTAL DESIGN

Model handoff and context transfer. For simplicity and controlled analysis, we study model collaboration through one-time handoffs, where an initial model begins a task and transfers control to another after several turns. The receiving model then continues until termination, without switching back to the source model. The total cost of a handoff run is the sum of the source model's cost before handoff and the receiving model's cost afterward. The receiving model inherits the workspace state, including all file modifications and artifacts produced by the source model. To transfer the accompanying execution history, we consider two approaches. 1) *In-context transfer* directly includes the pre-handoff conversation in the receiving model's context, preserving message roles, assistant responses, tool calls, and their corresponding tool outputs. When the source model uses a different response format, we convert its visible text and tool interactions into the receiving model's chat format. When the source model does not expose its chain-of-thought, we omit the `reasoning_content` field rather than reconstructing unavailable reasoning. 2) *Logfile transfer* instead writes the history to a `.jsonl` file inside the task environment, where the receiving model can inspect it using its tools. The file contains one JSON record per source-model turn, ordered chronologically. Each record includes the turn index, the original model response, and a list of actions. Each action records the original command and the tool output, including its return code, standard output, and standard

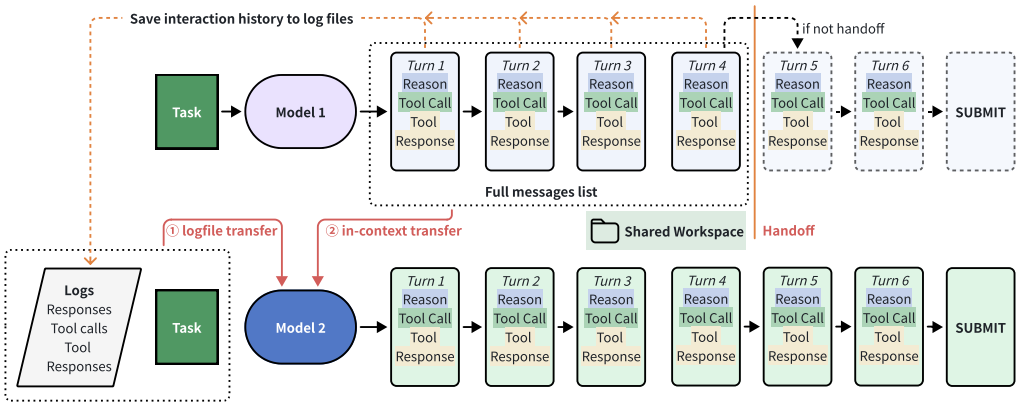

Figure 1: Overview of one-time model handoff and context transfer. The receiving model inherits the workspace at handoff and continues the task. Execution history is provided either as an inspectable logfile or directly in context.

error. Under *Logfile transfer*, the receiving model starts a new agent session with the original task instruction, the logfile path, and a textual schema describing the fields and structure of the log. The instruction specifies that records are ordered chronologically and directs the model to inspect the final records first to understand the latest task state. Figure 1 illustrates the handoff process and the two context-transfer approaches.

Benchmarks, Models, Harness and Settings We evaluate on Terminal-Bench 2.1 (Merrill et al., 2026) and DeepSWE v1.1 (Huang et al., 2026). Terminal-Bench covers diverse terminal-based workflows, including software engineering, scientific computing, and system administration, whereas DeepSWE focuses on challenging, long-horizon software engineering tasks in active open-source repositories, requiring substantial implementation work evaluated using program-based verifiers. For the analysis experiments, we primarily use GPT-5.6 Sol (OpenAI, 2026c), Kimi K3 (Moonshot AI, 2026), and GLM-5.3 (Z.ai, 2026) as capable models, with reasoning effort set to `max`. We use Qwen3.8-27B (Qwen Team, 2026c) as the efficient model, with reasoning effort set to `xhigh`, as it offers strong coding and reasoning capabilities at a scale suitable for deployment on local workstations or PCs. We adopt mini-SWE-agent v2.4.0 (Yang et al., 2024b), a minimal yet effective scaffold with a shell-based action interface and a linear interaction history. Its simple agent loop provides a common interface for comparing models and inspecting their trajectories, while capturing the basic interaction pattern of more complex harnesses. The scaffold is also used in official evaluations of frontier foundation models (DeepSeek, 2026; Anthropic, 2026a; Z.ai, 2026; Moonshot AI, 2026). For both benchmarks, we impose a two-hour agent execution timeout per task and allow up to 16K newly generated tokens per turn. We use the model providers' recommended sampling parameters where available and otherwise retain the serving defaults. For cloud models, we calculate cost from the token usage reported for each request and the provider's official token pricing, while assigning zero cost to locally deployable models under our accounting convention.

3 Understanding Model Handoffs in Agentic Tasks

3.1 Handoff Direction and Context Transfer

We first run each of the three capable models and the efficient model independently to obtain single-model baselines and execution trajectories. For a given trajectory, we replay its recorded actions in a task container to restore the workspace state after a selected turn, allowing a receiving model to continue from that checkpoint. We select handoff points at 30% and 75% of the source model's completed baseline trajectory to represent relatively early and late stages of execution. At each checkpoint, we evaluate both capable-to-efficient and efficient-to-capable handoffs using the in-context and logfile transfer methods described in Section 2.

Table 1: Context transfer across handoff directions at 30% and 75% of the source model's baseline trajectory. Each cell reports accuracy (%) / mean recorded API cost per task (USD) in the handoff runs. Bold marks strictly higher accuracy or lower cost between transfer methods at the same checkpoint. GPT, GLM, Kimi, and Qwen denote GPT-5.6 Sol, GLM-5.3, Kimi K3, and Qwen3.8-27B.

	Terminal-Bench 2.1				DeepSWE v1.1			
	30%		75%		30%		75%	
Handoff	In-context	Logfile	In-context	Logfile	In-context	Logfile	In-context	Logfile
Efficient → Capable								
Qwen → GLM	83.2 / 1.32	83.2 / **1.22**	68.5 / **1.19**	**79.8** / 1.26	51.3 / 4.04	**62.8 / 2.93**	38.9 / 2.61	**53.1 / 1.87**
Qwen → Kimi	79.8 / 2.91	**84.3 / 1.58**	70.8 / 3.31	**79.8 / 1.30**	47.8 / 7.77	**62.0 / 4.45**	37.2 / 5.76	**50.4 / 2.52**
Qwen → GPT	83.2 / 1.19	**89.9 / 0.87**	82.0 / 2.15	**85.4 / 0.85**	**73.5** / 6.20	70.8 / **5.00**	58.4 / 8.84	**67.3 / 5.74**
Capable → Efficient								
GLM → Qwen	**77.5** / 0.15	76.4 / 0.15	**83.2** / 0.64	79.8 / 0.64	**61.1** / 0.56	54.9 / 0.56	**62.8** / 2.07	59.3 / 2.07
Kimi → Qwen	**79.8** / 0.33	75.3 / 0.33	**87.6** / 1.06	82.0 / 1.06	**69.9** / 1.11	56.6 / 1.11	76.1 / 3.34	76.1 / 3.34
GPT → Qwen	64.0 / 0.24	**69.7** / 0.24	**83.2** / 0.70	77.5 / 0.70	54.9 / 1.25	**56.6** / 1.25	66.4 / 3.65	**67.3** / 3.65

The results in Table 1 show that the effect of context transfer depends on the handoff direction. For efficient-to-capable handoffs, logfile transfer generally matches or exceeds the accuracy of in-context transfer, while significantly reducing the cost in most cases. In the reverse direction, in-context transfer is more effective for GLM 5.3 and Kimi K3, achieving higher accuracy in most cases. GPT-5.6 Sol yields a mixed result: logfile transfer has higher accuracy at 30% on both benchmarks and at 75% on DeepSWE v1.1, whereas in-context transfer performs better at 75% on Terminal-Bench. We hypothesize that this difference may partly stem from the availability of reasoning traces: unlike GLM-5.3 and Kimi K3, GPT-5.6 Sol does not expose its full chain-of-thought, which may limit the reasoning context available to the receiving model under in-context transfer. Thus, we conclude that logfile transfer is more favorable for efficient-to-capable handoffs, while the preferred format for capable-to-efficient handoffs depends on the source model and checkpoint.

3.2 When Does Handoff Preserve Accuracy?

We next ask at what stages of task progress a handoff can preserve capable-only accuracy while substantially reducing cost. Building on the context-transfer comparison, we evaluate both handoff directions at 15%, 30%, 45%, 60%, 75%, and 90% of the source model's completed baseline trajectory. We use logfile transfer for efficient-to-capable handoffs and in-context transfer for GLM-5.3 and Kimi K3 handing off to Qwen 3.8 27B. For GPT-5.6 Sol handing off to Qwen 3.8 27B, we use logfile transfer, which achieves higher accuracy in most comparisons in Section 3.1. Figure 2 reports DeepSWE v1.1 accuracy and mean cost per task. The 0% endpoint corresponds to running the receiving model alone. Terminal-Bench 2.1 results are reported in Appendix B.

Efficient-to-capable handoffs. Figures 2(a) and 2(b) show that starting with the efficient model does not consistently preserve capable-only accuracy while substantially reducing cost. On DeepSWE v1.1, later handoffs to GLM-5.3 and Kimi K3 generally reduce cost but also lose accuracy. Handoffs to GPT-5.6 Sol can retain comparable accuracy, yet offer limited savings: at 30% progress, accuracy reaches 70.80%, compared with 68.10% for GPT-5.6 Sol alone, while cost falls by only 6.1%. Thus, delaying the capable model's involvement does not provide a consistent way to achieve substantial savings without sacrificing accuracy. These results also imply that fine-grained or per-turn routing may face steep hurdles, as frequent switching can amplify coordination and recovery overheads.

Capable-to-efficient handoffs. Figures 2(c) and 2(d) show more favorable trade-offs in the reverse direction: at suitable stages of progress, capable-to-efficient handoffs can approach capable-only accuracy while substantially reducing cost. With Kimi K3 as the source model, handing off at 45% reaches 72.57% accuracy, close to the capable-only baseline of 73.45%, while reducing cost by 61.5%. With GPT-5.6 Sol, logfile transfer at 60% reaches 67.26% accuracy versus 68.10% for this capable model alone, with 47.9% lower cost. GLM-5.3 can hand off at 30% and attain 61.1% accuracy, matching its capable-only baseline, while reducing cost by 82.4%. These results suggest that progress made by a capable model can enable an efficient model to complete the remaining work.

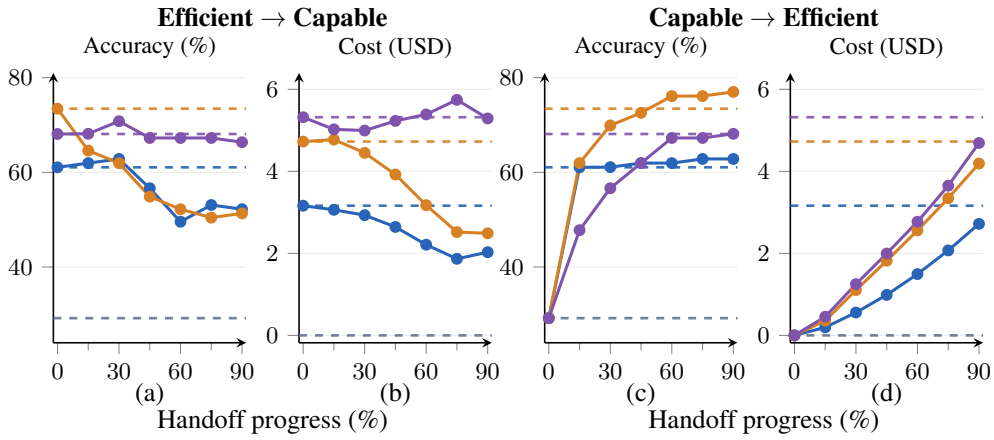

Figure 2: Accuracy and mean API cost across handoff progress on DeepSWE v1.1. Colors identify the capable model: — GLM-5.3, — Kimi K3, and — GPT-5.6 Sol; the efficient model is Qwen 3.8 27B. Solid curves show handoffs using logfile transfer in (a–b); in (c–d), GLM-5.3 and Kimi K3 use in-context transfer, while GPT-5.6 Sol uses logfile transfer. Horizontal dashed lines (- - - - - -) show the corresponding capable-only baselines; - - denotes Qwen-only execution. Progress is relative to the source model's completed baseline trajectory; 0% denotes receiving-model-only execution.

3.3 Patterns of Successful Handoffs

The checkpoint experiments show that handoff timing matters, but do not explain what task progress enables an efficient model to take over successfully. To investigate this question, we analyze trajectories where Qwen 3.8 27B fails after an earlier handoff but succeeds after the next one, comparing the capable model's intervening actions and tool feedback to identify recurring patterns. Figure 3 presents three illustrative cases across two benchmarks and two capable models, all of which fail under Qwen-only execution and succeed under capable-only execution. The key changes include (a) fixing a quoted-path handling issue, (b) validating the coordinate transformation, and (c) localizing an encoding fault that helps Qwen reimplement a still-incorrect encoder.

The analysis suggests three recurring signs of handoff readiness: 1) a substantive implementation, runnable prototype, or task-required artifact is available; 2) actual execution, testing, or inspection has provided task-relevant feedback; and 3) the remaining work has become more focused, often involving localized diagnosis, repair, validation, or final-output preparation. Crucially, this feedback need not be 100% positive; even a failed execution is informative if it narrows down the unresolved problem, as demonstrated in the compressor case. These observations suggest that readiness depends on both the progress already established and the nature of the remaining work, rather than elapsed turns alone.

4 AgentBaton

The patterns identified in Section 3.3 motivate a natural question: can we train a lightweight model to recognize suitable handoff points during a capable model's execution? Such a detector would enable the system to transfer control to an efficient model when the observed progress indicates that it is ready to take over.

Table 2: Prompt-based handoff to Qwen 3.8 27B on DeepSWE v1.1. Handoffs reports the number of tasks with a recorded handoff; cost is mean API cost/task.

Capable model	Handoffs	Acc. (%)	Cost ($)
GLM-5.3	6/113	54.9	2.54
Kimi K3	0/113	64.6	4.89
GPT-5.6 Sol	113/113	62.8	3.75

4.1 Prompt-based handoff

Before training a detector, we test whether the capable model can decide when to hand off through prompting alone. We prompt the capable

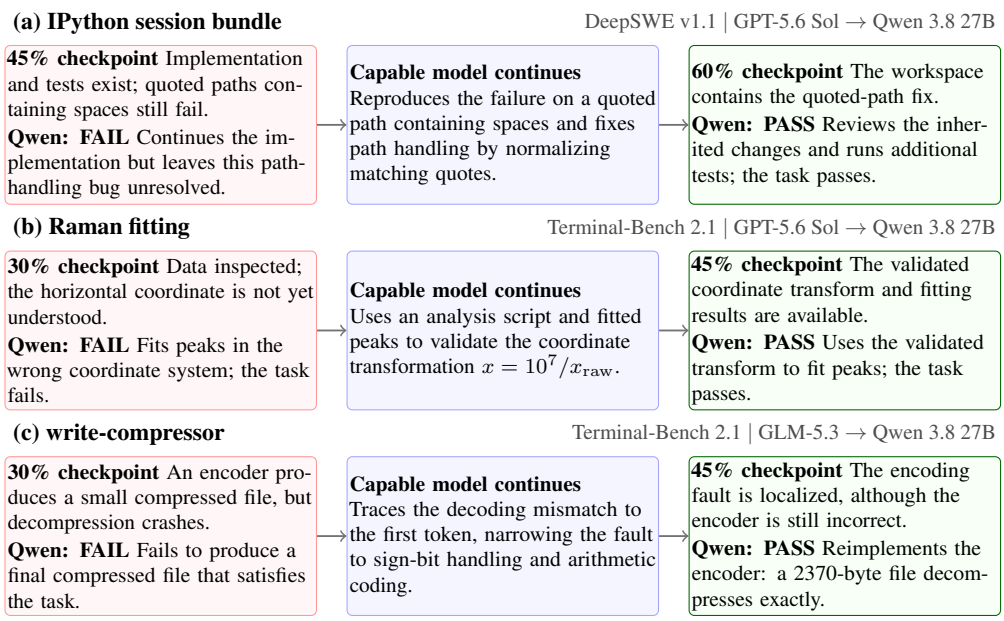

Figure 3: Unsuccessful and successful capable-to-efficient handoffs. Each row shows an earlier checkpoint, the capable model's intervening progress, and the next checkpoint. Left and right boxes report separate Qwen 3.8 27B continuations. Percentages refer to the capable-only trajectory.

model to transfer control when it recognizes the handoff-readiness patterns identified in Section 3.3. Table 2 shows that this behavior varies substantially across models: Kimi K3 never hands off, while GLM-5.3 does so on only 6 of 113 tasks. Thus, a handoff instruction does not reliably induce models to relinquish control, limiting opportunities to reduce capable-model usage. GPT-5.6 Sol does hand off on every task, but its mean cost remains $3.75, compared with $5.32 for capable-only execution. Even consistent handoff behavior therefore does not ensure a favorable accuracy–cost trade-off. These results motivate a dedicated detector that explicitly evaluates handoff readiness, rather than relying on the executing model to decide when to transfer control.

4.2 Training AgentBaton

Data collection and annotation. We collect agent trajectories on a randomly sampled subset of 2,000 tasks from SWE-rebench V2 (Badertdinov et al., 2026). For each task, we randomly select two distinct capable models from a pool of five: GPT-6 Astra (OpenAI, 2026d), GPT-5.6 Sol (OpenAI, 2026c), Claude Opus 5 (Anthropic, 2026a), Kimi K3 (Moonshot AI, 2026), and GLM-5.3 (Z.ai, 2026). We use `medium` reasoning effort for GPT-6 Astra and `max` for the other models. Each selected model independently attempts the task using mini-swe-agent, yielding 4,000 trajectories. We use Codex[1] to annotate suitable handoff points in these trajectories. The annotation criteria generally follow the qualitative patterns identified in Section 3.3: a substantive implementation or task-required artifact exists, task-relevant execution or inspection provides concrete feedback, and the remaining work is primarily localized diagnosis, repair, or validation. Codex reviews each trajectory retrospectively to better locate the earliest suitable handoff point. Trajectories without a clear transition receive a separate no-clear-handoff label. Appendix C provides the annotation protocol, including evidence restrictions and turn indexing.

Training details. We initialize AgentBaton from Qwen 3.5 2B (Qwen Team, 2026a) for low-overhead detection at each turn.[2] The input is a multi-turn conversation consisting of a system instruction that guides pattern detection, the original task, and the capable model's actions and tool observations, interleaved with previous handoff decisions. For a trajectory whose annotated

[1] https://openai.com/codex/

[2] We also train a Qwen 3.5 9B (Qwen Team, 2026b) variant but observe no clear improvement.

handoff occurs at turn h, we supervise every decision up to and including turn h: the model is trained to output the JSON object {"judgement":"HANDOFF_NOT_READY"} for turns before h and {"judgement":"HANDOFF_READY"} at turn h. We also train on all subsequent turns. Each such turn yields an additional example in which the target response for the current turn is set to {"judgement":"HANDOFF_READY"}. All preceding decision responses are set to {"judgement":"HANDOFF_NOT_READY"} and excluded from the loss calculation. These additional positive examples help balance the decision labels and simulate missed handoff opportunities, encouraging the detector to trigger a handoff at a later turn even if it previously failed to do so. We train for two epochs with a maximum sequence length of 32K tokens. We split and truncate overlong sequences and pack shorter ones together to reduce padding and improve computational efficiency. We train with NeMo Automodel (NVIDIA-Nemo, 2026) on 16 NVIDIA H100 GPUs using FSDP2, bfloat16 precision, and activation checkpointing. The global batch size is 16 packed sequences, with one sequence per GPU and no gradient accumulation. We use AdamW with a learning rate of 10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and weight decay 0.1.

Online Handoff Detection At inference time, the capable model begins the task. After each turn, AGENTBATON processes the updated trajectory and predicts whether a suitable handoff point has been reached. When the context exceeds this limit, we discard the oldest half of the historical turns while always retaining the system instruction and original task. A HANDOFF_NOT_READY decision leaves control with the capable model; the first HANDOFF_READY decision transfers control to the efficient model, which inherits the current workspace and continues until termination using the context-transfer method selected for that model pair. Figure 4 illustrates this process. Across 360 evaluations on real inputs ranging from 2K to 121K tokens, AGENTBATON achieves a median latency of 338 ms and a 95th-percentile latency of 1.11 s on a single NVIDIA H100 GPU, demonstrating low latency.

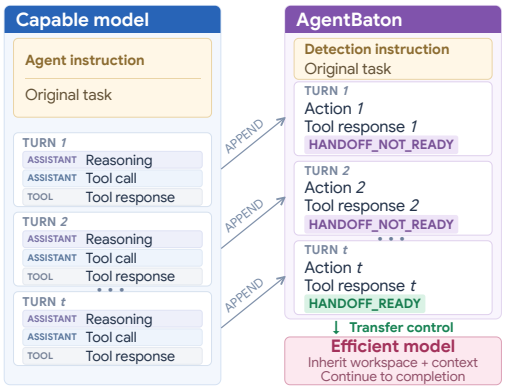

Figure 4: Online detection with separate, growing message histories. Each arrow appends the latest action and tool response to AGENTBATON, which retains its previous decisions.

4.3 EXPERIMENTAL RESULTS

We further extend the model pool in Section 3 with two additional capable models, Claude Opus 5 (Anthropic, 2026a) and Gemini 3.8 Flash (Google DeepMind, 2026), and another efficient model, GPT-5.6 Luna (OpenAI, 2026c). We also evaluate GPT-5.6 Luna as a source model handing off to Qwen 3.8 27B, testing whether handoff detection remains useful when the source is itself a lower-cost model. We evaluate both in-context and logfile transfer for each model pair. Table 3 uses a fixed context-transfer method for each source model across both benchmarks and receiving models: logfile transfer for GPT-5.6 Sol and GPT-5.6 Luna, and in-context transfer for all other source models. Complete results for both transfer methods are provided in Appendix D.

The results show that AGENTBATON detects handoff points at an average of 26.6–41.1% of the source model's trajectory on DeepSWE, compared with 55.6–68.9% on Terminal-Bench, indicating that detected handoff points occur at different stages across benchmarks. AGENTBATON substantially reduces API cost, with accuracy preservation varying across model pairs and benchmarks: 12 of the 22 (model-pair, benchmark) settings either improve accuracy or incur a relative loss of less than 5%. For example, on DeepSWE, Kimi K3 handing off to Qwen 3.8 27B achieves 72.6% accuracy compared with 73.5% for capable-only execution, while reducing cost by 72.9%. A stronger efficient model can further improve the outcome: with GPT-5.6 Luna, four of the five capable models exceed their capable-only accuracy on DeepSWE, while GPT-5.6 Sol incurs only a 1.2% relative loss. These configurations reduce cost by 66.6–73.9%; for instance, Opus-to-Luna improves accuracy from 67.3% to 76.1% while reducing cost by 70.5%. Even when the source model is already efficient,

Table 3: AGENTBATON results. We use logfile transfer when the source model is GPT-5.6 Sol or GPT-5.6 Luna, and in-context transfer for all other source models. Subscripts give relative changes from capable-only: green is better, red worse. Handoff is the mean detected progress (%), shared across receiving models for a given source. Dashes denote single-model runs.

		DeepSWE v1.1			Terminal-Bench 2.1		
Capable model	Efficient model	Handoff (%)	Acc. (%) ↑	Cost ($) ↓	Handoff (%)	Acc. (%) ↑	Cost ($) ↓
—	Qwen 3.8 27B	—	29.2	0.00	—	69.7	0.00
—	GPT-5.6 Luna	—	60.2	0.40	—	82.2	0.07
GPT-5.6 Sol	—	—	68.1	5.32	—	87.6	0.97
	Qwen 3.8 27B	26.6	60.2$_{-11.6\%}$	1.02$_{-80.8\%}$	58.4	82.0$_{-6.4\%}$	0.43$_{-55.7\%}$
	GPT-5.6 Luna		67.3$_{-1.2\%}$	1.39$_{-73.9\%}$		82.0$_{-6.4\%}$	0.47$_{-51.5\%}$
Claude Opus 5	—	—	67.3	9.91	—	84.3	2.83
	Qwen 3.8 27B	36.6	61.9$_{-8.0\%}$	2.61$_{-73.7\%}$	55.6	87.6$_{+3.9\%}$	1.13$_{-60.1\%}$
	GPT-5.6 Luna		76.1$_{+13.1\%}$	2.92$_{-70.5\%}$		86.5$_{+2.6\%}$	1.17$_{-58.7\%}$
GLM-5.3	—	—	61.1	3.16	—	86.5	1.02
	Qwen 3.8 27B	28.6	58.4$_{-4.4\%}$	0.49$_{-84.5\%}$	55.7	80.9$_{-6.5\%}$	0.27$_{-73.5\%}$
	GPT-5.6 Luna		66.4$_{+8.7\%}$	0.83$_{-73.7\%}$		79.8$_{-7.7\%}$	0.33$_{-67.6\%}$
Kimi K3	—	—	73.5	4.73	—	86.5	1.50
	Qwen 3.8 27B	35.3	72.6$_{-1.2\%}$	1.28$_{-72.9\%}$	64.0	86.5$_{0.0\%}$	0.57$_{-62.0\%}$
	GPT-5.6 Luna		77.0$_{+4.8\%}$	1.58$_{-66.6\%}$		80.9$_{-6.5\%}$	0.62$_{-58.7\%}$
Gemini 3.8 Flash	—	—	65.5	6.15	—	86.5	1.02
	Qwen 3.8 27B	41.1	61.9$_{-5.5\%}$	1.21$_{-80.3\%}$	68.9	86.5$_{0.0\%}$	0.51$_{-50.0\%}$
	GPT-5.6 Luna		73.5$_{+12.2\%}$	1.71$_{-72.2\%}$		85.4$_{-1.3\%}$	0.56$_{-45.1\%}$
GPT-5.6 Luna	Qwen 3.8 27B	32.6	55.8$_{-7.3\%}$	0.10$_{-75.0\%}$	64.4	77.5$_{-5.7\%}$	0.04$_{-42.9\%}$

Luna-to-Qwen reduces API cost by 75.0% on DeepSWE and 42.9% on Terminal-Bench, with relative accuracy losses of 7.3% and 5.7% respectively, indicating the versatility of AGENTBATON.

4.4 COMPARISON WITH ROUTING AND HANDOFF BASELINES

To put these results in context, we compare AGENTBATON with query-level and per-turn routing as alternative ways. We first test query-level routing by fine-tuning Qwen 3.5 2B to predict whether Qwen 3.8 27B can solve a task from its initial description, using success labels from 2,000 SWE-rebench V2 rollouts with balanced positive and negative examples. Despite 99.55% acc. on the training set, the router selects Qwen for only one out of all tasks, thus it effectively reduces to capable-only execution and provides little cost savings.

Table 4: AGENTBATON versus routing and fixed handoff baselines with Qwen 3.8 27B. Fixed thresholds use the mean progress or turn under AGENTBATON for each setting. Accuracy is in percent; cost is mean API cost per task.

	DeepSWE v1.1		Terminal Bench 2.1	
Method	Acc. (%)	Cost ($)	Acc. (%)	Cost ($)
Capable: GLM 5.3; Efficient: Qwen 3.8 27B				
MTRouter	58.4	3.67	80.9	1.35
Fixed progress	59.3	0.52	78.7	0.38
Fixed turn	54.9	0.48	76.4	0.18
AGENTBATON	58.4	0.49	80.9	0.27
Capable: Kimi K3; Efficient: Qwen 3.8 27B				
MTRouter	66.4	4.37	85.4	1.07
Fixed progress	74.3	1.28	86.5	0.83
Fixed turn	69.9	1.15	82.0	0.37
AGENTBATON	72.6	1.28	86.5	0.57

We then reproduce MTRouter (Zhang et al., 2026b) with the same model pairs to investigate whether a more fine-grained, model-specific per-turn routing strategy can achieve better accuracy–cost trade-offs. As shown in Table 4, AGENTBATON reduces API cost by 46.7–86.6% relative to MTRouter, while matching or improving accuracy in all four settings. These results indicate that our learned one-time handoff strategy achieves even better accuracy–cost trade-offs than per-turn routing in the evaluated settings.

We also compare AGENTBATON with fixed-progress and fixed-turn handoffs, using its mean handoff progress and mean handoff turn for each benchmark and model pair as the respective thresholds. Fixed-progress handoff has access to future information, i.e., the source model's final trajectory length, that is unavailable during online execution. Table 4 reveals that despite this advantage, its accuracy remains close to AGENTBATON, while AGENTBATON incurs equal or lower API cost.

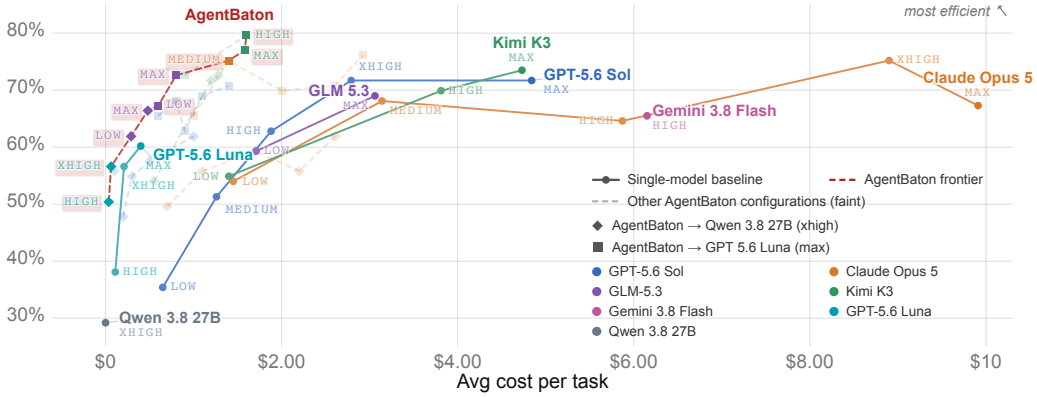

Figure 5: Accuracy–cost trade-offs across reasoning-effort settings on DeepSWE v1.1. Solid lines and circles denote single-model baselines. For AGENTBATON, diamonds denote handoffs to Qwen 3.8 27B and squares denote handoffs to GPT-5.6 Luna; colors identify the source model, as listed in the legend. The dark red dashed line connects the empirical Pareto frontier among evaluated AGENTBATON configurations, whose effort labels are highlighted. For example, the purple ◆ labeled `MAX` denotes GLM-5.3 at `max` effort handing off to Qwen 3.8 27B at `xhigh` effort; the green ■ labeled `HIGH` denotes Kimi K3 at `high` effort handing off to GPT-5.6 Luna at `max` effort.

Compared with fixed-turn handoff, AGENTBATON achieves consistently higher accuracy. These results show that online detection can achieve accuracy comparable to a fixed-progress baseline with access to future information, while preserving accuracy better than a fixed turn threshold.

4.5 PUSHING THE ACCURACY–COST PARETO FRONTIER

Finally, we examine whether AGENTBATON can improve the accuracy–cost trade-offs obtained by varying a model's reasoning effort. On DeepSWE v1.1, we sweep `low`, `medium`, `high`, `xhigh`, and `max` for GPT-5.6 Sol and Claude Opus 5; `low` and `max` for GLM-5.3; `low`, `high`, and `max` for Kimi K3; and `high`, `xhigh`, and `max` for GPT-5.6 Luna. For each source configuration, we use AGENTBATON to detect a handoff point and continue with Qwen 3.8 27B. We additionally evaluate continuations with GPT-5.6 Luna for the first four source models. We use logfile transfer for handoffs from GPT-5.6 Sol and GPT-5.6 Luna, and in-context transfer for all other source models. Figure 5 compares these configurations with single-model baselines, and Appendix E provides the complete numerical results. The results show that AGENTBATON extends the empirical accuracy–cost frontier beyond the evaluated single-model configurations across a range of budgets. At the low-cost end, Luna-to-Qwen with `xhigh` effort matches Luna's standalone accuracy of 56.6% while reducing mean API cost from $0.21 to approximately $0.06 per task. At a higher budget, GLM-to-Luna with `max` effort achieves 72.6% accuracy at $0.80 per task, compared with 69.0% at $3.06 for GLM alone. Kimi-to-Luna with `high` effort reaches 79.6% accuracy at $1.60 per task, exceeding the highest single-model accuracy in the sweep, 75.2% from Opus with `xhigh` effort at $8.90. Different source models, effort settings, and receiving models contribute to the frontier, suggesting that reasoning effort and model handoff offer complementary ways to allocate computation over a task.

5 CONCLUSION

We study how capable and efficient models can collaborate within a single long-horizon agentic task through a one-time handoff. Our experiments show that capable-to-efficient handoffs offer favorable accuracy–cost trade-offs, particularly once the capable model has established an implementation, gathered feedback, and narrowed the remaining work to localized refinement. Motivated by these findings, we introduce AGENTBATON, a lightweight detector that dynamically identifies suitable handoff points. Across DeepSWE v1.1 and Terminal-Bench 2.1, AGENTBATON substantially reduces API cost while largely preserving, and occasionally improving, task accuracy. Further reasoning-effort sweeps demonstrate that such handoffs can extend the empirical Pareto frontier beyond single-model configurations, highlighting task progress as an effective signal for dynamic capability allocation.

AI USE STATEMENT

We used generative AI to assist with manuscript drafting and editing and literature discovery. We used Codex for code implementation, automated experiment execution, and data analysis, including qualitative analysis of agent trajectories to identify recurring handoff patterns. We also used Codex to generate handoff-point annotations that served as supervision for training AGENTBATON, as described in the paper.

ETHICS STATEMENT

This work studies model collaboration on software engineering and terminal-based benchmarks. More efficient agents may support beneficial applications, but can also lower the cost of harmful automation; benchmark performance alone does not establish safety in deployment. Deployment should retain appropriate access controls and validation of agent actions and outputs.

REFERENCES

Yash Akhauri, Anthony Fei, Chi-Chih Chang, Ahmed F AbouElhamayed, Yueying Li, and Mohamed S Abdelfattah. Splitreason: Learning to offload reasoning. *arXiv preprint arXiv:2504.16379*, 2025.

Anthropic. Introducing Claude Opus 5, 2026a. URL https://www.anthropic.com/news/claude-opus-5.

Anthropic. Introducing Claude Fable 5.1 and Claude Mythos 5.1, 2026b. URL https://www.anthropic.com/claude-fable-and-mythos-5-1.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. SWE-rebench v2: Language-agnostic SWE task collection at scale. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=UCAda9kS57.

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*, 2023.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems*, 37:66305–66328, 2024.

DeepSeek. Introducing DeepSeek-V4.1-Flash: smarter, faster, more efficient., 2026. URL https://www.deepseek.com/en/news/deepseek-v4-1-flash/.

Jasper Dekoninck, Maximilian Baader, and Martin Vechev. A unified approach to routing and cascading for llms. *arXiv preprint arXiv:2410.10347*, 2024.

Roy Ganz, Mor Shpigel Nacson, Adi Kalyanpur, and Ron Litman. The handoff tax: Continuing non-native trajectories in LLM agents. *arXiv preprint arXiv:2608.24358*, 2026. URL https://arxiv.org/abs/2608.24358.

Google DeepMind. Introducing Gemini 3.8 Flash and 3.8 Flash Cyber, 2026. URL https://blog.google/innovation-and-ai/models-and-research/gemini-models/3-8-flash-and-3-8-flash-cyber/.

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. *arXiv preprint arXiv:2403.12031*, 2024.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks, 2026. URL https://github.com/datacurve-ai/deep-swe.

Sangmook Lee, Dohyung Kim, Hyukhun Koh, Nakyeong Yang, and Kyomin Jung. Confidence-guided stepwise model routing for cost-efficient reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31483–31491, 2026.

Ji'an Lei and Jian Huang. Tacit-switch: Cost-aware model escalation for llm agents from censored supervision. *arXiv preprint arXiv:2608.27911*, 2026.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Moonshot AI. Kimi K3: Open Frontier Intelligence, 2026. URL https://www.kimi.ai/blog/kimi-k3.

NVIDIA-Nemo. NeMo AutoModel: DTensor-native SPMD library for scalable and efficient training. https://github.com/NVIDIA-NeMo/Automodel, 2026. GitHub repository.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026a. URL https://openai.com/index/gpt-6-astra/.

OpenAI. Introducing GPT-6 Sol and Luna, 2026b. URL https://openai.com/index/introducing-gpt-6-sol-and-luna/.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, 2026c. URL https://openai.com/index/gpt-5-6/.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026d. URL https://openai.com/index/gpt-6-astra/.

Zhihong Pan, Kai Zhang, Yuze Zhao, and Yupeng Han. Route to reason: Adaptive routing for llm and reasoning strategy selection. *arXiv preprint arXiv:2505.19435*, 2025.

Qwen Team. Qwen3.5-2B, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-2B.

Qwen Team. Qwen3.5-9B, 2026b. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Qwen Team. Qwen3.8-27B, 2026c. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Seongho Son, Sangwoong Yoon, Jiahua Tang, Shuhan Wang, Lorenz Wolf, and Ilija Bogunovic. Swe-router: Routing in multi-turn agentic software engineering tasks. *arXiv preprint arXiv:2607.00053*, 2026.

Jiwon Song, Yoongon Kim, and Jae-Joon Kim. Relaygen: Intra-generation model switching for efficient reasoning. *arXiv preprint arXiv:2602.06454*, 2026.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Jiayu Wang, Yifei Ming, Zixuan Ke, Shafiq Joty, Aws Albarghouthi, and Frederic Sala. Skillorchestra: Learning to route agents via skill transfer. *arXiv preprint arXiv:2602.19672*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024a.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://arxiv.org/abs/2405.15793.

Wang Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. *arXiv preprint arXiv:2504.12329*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Z.ai. GLM-5.3: Frontier Coding with Emergent Cyber Capabilities, 2026. URL https://z.ai/blog/glm-5.3.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 141233–141265, 2026a.

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. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44206–44226, 2026b.

Pengfei Zhou, Zhiwei Tang, Yixing Ma, Jiasheng Tang, Yizeng Han, Zhenglin Wan, Fanqing Meng, Wei Wang, Bohan Zhuang, Wangbo Zhao, et al. Agent-as-a-router: Agentic model routing for coding tasks. *arXiv preprint arXiv:2606.22902*, 2026.

A Related work

Request-level routing and cascades. Prior work optimizes inference allocation at the request level. For instance, FrugalGPT (Chen et al., 2023) and Dekoninck et al. (2024) explore cost-aware cascading pipelines, while RouteLLM (Ong et al., 2025), RouterDC (Chen et al., 2024), and Route to Reason (Pan et al., 2025) learn query-level model selection, compatibility, or reasoning strategies, supported by unified evaluation benchmarks (Hu et al., 2024). These approaches primarily address individual requests. In interactive agentic tasks, an agent's actions modify the workspace and accumulate interaction history, changing the state inherited by a subsequent model. Consequently, determining which model can resolve this evolving state requires accounting for task progress beyond the original request.

Multi-round routing and orchestration. Beyond single-turn decisions, multi-round orchestration architectures dynamically coordinate model and tool execution. Frameworks such as Router-R1 (Zhang et al., 2026a), ToolOrchestra (Su et al., 2025), and SkillOrchestra (Wang et al., 2026) learn competence, invocation patterns, or skill-specific costs from execution feedback. Similarly, MTRouter (Zhang et al., 2026b) uses interaction history for turn-level routing, while Agent-as-a-Router (Zhou et al., 2026) draws on verified experience across a stream of coding tasks. These approaches select among candidate models or tools, sometimes using model descriptions to support generalization to unseen candidates, as demonstrated by Router-R1 and ToolOrchestra. AgentBaton separates handoff timing from model selection: for a given capable–efficient model pair, a lightweight learned detector assesses task progress after each turn and decides whether to transfer control. We evaluate the same detector across multiple model pairs, with the receiving model continuing until termination after a handoff.

Model handoffs and trajectory transfer. A closely related line of research uses execution feedback to inform model transitions. SWE-Router (Son et al., 2026) uses an exploratory trajectory to inform escalation, while TACIT-Switch (Lei & Huang, 2026) learns permanent small-to-large handoffs based on accumulated risk signals. The Handoff Tax (Ganz et al., 2026) studies how handoff direction, timing, and trajectory-transfer interfaces affect coding-agent performance, finding direction-dependent accuracy–cost trade-offs. Its experiments use predefined switch points to isolate the effects of handoff and inherited history. Our analysis likewise examines handoff direction and context transfer, and further motivates a learned policy that identifies suitable capable-to-efficient handoff points online based on recurring patterns observed in agent trajectories.

Switching within a reasoning output. Finally, several approaches perform fine-grained handoffs within a single reasoning trajectory. STEER (Lee et al., 2026) and SplitReason (Akhauri et al., 2025) offload reasoning steps based on confidence or learned difficulty, while Speculative Thinking (Yang et al., 2025) delegates reflective reasoning to larger models. Most relevantly, RelayGen (Song et al., 2026) transfers simpler generation segments from a strong model to a smaller one via model-specific cues without router training. Our handoffs occur between complete agent turns, involving tool use, a shared workspace, and persistent changes to the task state. The receiving model inherits this workspace and accesses prior interaction history through in-context or logfile transfer, then continues acting until termination. Our focus is on how task progress and context transfer support continuation across models.

B Handoff progress on Terminal-Bench

Figure 6 reports the Terminal-Bench 2.1 results using the same handoff checkpoints as the DeepSWE experiments in the main text. As in the main text, we use logfile transfer for GPT-5.6 Sol handing off to Qwen 3.8 27B. For efficient-to-capable handoffs, accuracy and cost fluctuate across checkpoints, and Qwen-to-GLM handoffs cost more than GLM alone throughout the sweep. Capable-to-efficient handoffs offer favorable trade-offs at some checkpoints: Kimi K3 handing off at 75% reaches 87.64% accuracy versus 86.52% for Kimi alone, with 29.4% lower cost. However, the results are less consistent across models and checkpoints than on DeepSWE, and later handoffs do not monotonically improve accuracy.

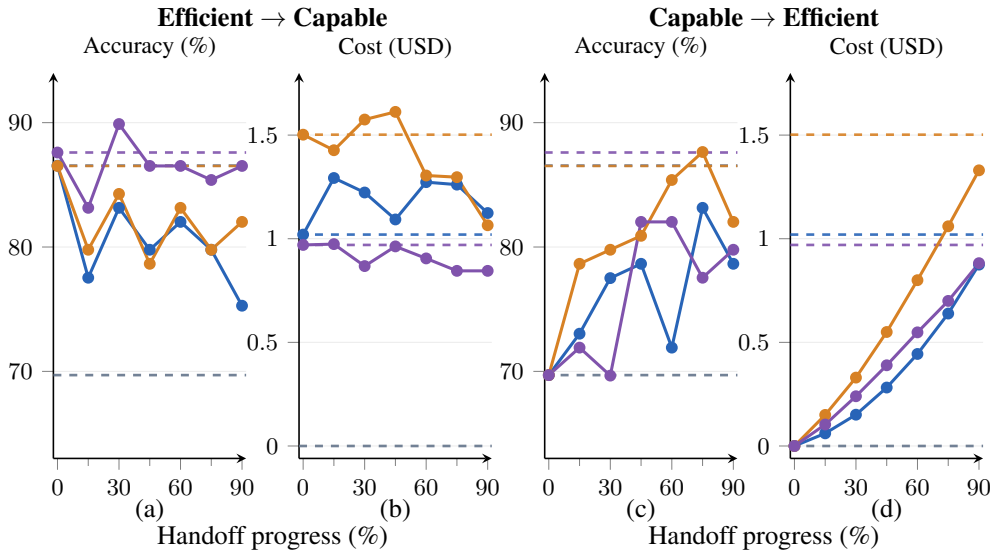

Figure 6: Accuracy and mean API cost across handoff progress on Terminal-Bench 2.1. Colors identify the capable model: — GLM-5.3, — Kimi K3, and — GPT-5.6 Sol; the efficient model is Qwen 3.8 27B. Solid curves show handoffs using logfile transfer in (a–b); in (c–d), GLM-5.3 and Kimi K3 use in-context transfer, while GPT-5.6 Sol uses logfile transfer. Horizontal dashed lines (- - - - - -) show the corresponding capable-only baselines; - - denotes Qwen-only execution. Progress is relative to the source model's completed baseline trajectory; 0% denotes receiving-model-only execution.

C HANDOFF-POINT ANNOTATION PROTOCOL

We use Codex to annotate the earliest suitable handoff point in each collected trajectory. The protocol follows the qualitative patterns in Section 3.3. Codex reviews each trajectory retrospectively to better locate the earliest suitable handoff point.

Evidence available to the annotator. The annotator reads the task description and the chronological sequence of tool calls and their responses. Private reasoning text, post-trajectory verifier outputs, final rewards, and outcomes of receiving-model continuations are excluded as annotation evidence.

Readiness criteria. The annotator selects the earliest turn at which the evidence establishes all three conditions:

1. **A substantive artifact exists.** There is an implementation, runnable prototype, or task-required artifact. Investigation, a plan, or an unexecuted code sketch alone is insufficient.
2. **The artifact has received real feedback.** It has been compiled, executed, tested, or inspected by a task-relevant checker, with an observed response. Issuing a command does not establish that it succeeded. A failed check can qualify when it exposes a localized problem; an environment failure that prevents the artifact from being exercised does not by itself establish this condition.
3. **The remaining work has become localized.** The evidence indicates that the work ahead mainly concerns diagnosis, repair, hardening, validation, or final-output preparation, rather than selecting the core approach or implementing the main solution from scratch.

These conditions do not require all tests to pass. Conversely, a passing check on an unrelated component or a substitute that bypasses the central requested behavior is insufficient. The annotator considers what the observed check actually establishes about the task-required artifact.

Table 5: Complete AGENTBATON results for both context-transfer methods. Each row reports accuracy and mean API cost per task from the same configuration.

			DeepSWE v1.1		Terminal-Bench 2.1	
Capable model	Efficient model	Transfer	Acc. (%)	Cost ($)	Acc. (%)	Cost ($)
GPT-5.6 Sol	Qwen 3.8 27B	In-context	54.9	1.03	85.4	0.43
		Logfile	60.2	1.02	82.0	0.43
	GPT-5.6 Luna	In-context	61.9	1.27	84.3	0.47
		Logfile	67.3	1.39	82.0	0.47
Claude Opus 5	Qwen 3.8 27B	In-context	61.9	2.61	87.6	1.13
		Logfile	61.1	2.44	87.6	1.06
	GPT-5.6 Luna	In-context	76.1	2.92	86.5	1.17
		Logfile	70.8	2.99	85.4	1.19
GLM-5.3	Qwen 3.8 27B	In-context	58.4	0.49	80.9	0.27
		Logfile	53.1	0.49	78.7	0.26
	GPT-5.6 Luna	In-context	66.4	0.83	79.8	0.33
		Logfile	67.3	0.87	85.4	0.34
Kimi K3	Qwen 3.8 27B	In-context	72.6	1.28	86.5	0.57
		Logfile	65.5	1.19	84.3	0.53
	GPT-5.6 Luna	In-context	77.0	1.58	80.9	0.62
		Logfile	74.3	1.64	85.4	0.61
Gemini 3.8 Flash	Qwen 3.8 27B	In-context	61.9	1.21	86.5	0.51
		Logfile	54.0	1.20	80.9	0.51
	GPT-5.6 Luna	In-context	73.5	1.71	85.4	0.56
		Logfile	68.1	1.64	85.4	0.56
GPT-5.6 Luna	Qwen 3.8 27B	In-context	54.9	0.10	74.2	0.04
		Logfile	55.8	0.10	77.5	0.04

Selecting the first point and handling uncertainty. The label records the first qualifying point, not the final successful test or submission. If later work introduces a regression or substantially rewrites the implementation, the earlier point is retained and the instability is noted. If the conditions are never jointly established, the trajectory is labeled `no_clear_handoff_point`, with no handoff index. Missing or unreadable trajectories are recorded separately and are not assigned this label. Readable failed or interrupted trajectories remain eligible for annotation. The provenance of an implementation does not by itself determine readiness; when relevant, it is documented as a caveat.

Turn indexing and annotation record. A turn consists of one assistant step containing one or more tool calls together with their responses. Both the handoff index and evidence indices are zero-based within these tool-bearing steps, rather than indices over all conversation messages. We also preserve each step's original identifier to support traceability to the source trajectory. Each record includes the task and source model, the actual trajectory path, the total number of tool-bearing turns, the readiness status, the handoff index and source step identifier, supporting tool evidence, a brief rationale, a confidence level, and any caveats. For trajectories without a clear handoff point, the handoff fields are null.

D COMPLETE AGENTBATON RESULTS

Table 5 reports both in-context and logfile transfer for all five capable models and both efficient models, as well as GPT-5.6 Luna handing off to Qwen 3.8 27B. The main-text table uses logfile transfer for GPT-5.6 Sol and GPT-5.6 Luna as source models, and in-context transfer for all other source models. Single-model baselines are reported in Table 3.

Table 6: Complete numerical results for the reasoning-effort sweep on DeepSWE v1.1 in Figure 5. Effort refers to the source model. Accuracy is in percent and cost is mean API cost in USD per task. Dashes indicate configurations not evaluated in this sweep.

		Single model		AgentBaton → Qwen		AgentBaton → Luna	
Source model	Effort	Acc.	Cost	Acc.	Cost	Acc.	Cost
GPT-5.6 Sol	`low`	35.4	0.65	47.8	0.20	65.5	0.60
	`medium`	51.3	1.26	54.9	0.30	68.1	0.80
	`high`	62.8	1.88	57.5	0.50	62.8	0.90
	`xhigh`	71.7	2.79	59.3	0.70	69.0	1.10
	`max`	71.7	4.84	61.9	1.00	70.8	1.40
Claude Opus 5	`low`	54.0	1.45	49.6	0.70	65.5	1.00
	`medium`	68.1	3.14	55.8	1.10	75.2	1.40
	`high`	64.6	5.87	59.3	1.70	69.9	2.00
	`xhigh`	75.2	8.90	55.8	2.20	70.8	2.60
	`max`	67.3	9.91	61.9	2.61	76.1	2.92
GLM-5.3	`low`	59.3	1.71	61.9	0.29	67.3	0.60
	`max`	69.0	3.06	66.4	0.48	72.6	0.80
Kimi K3	`low`	54.9	1.40	54.0	0.55	72.6	0.90
	`high`	69.9	3.81	71.7	1.20	79.6	1.60
	`max`	73.5	4.73	72.6	1.28	77.0	1.58
Gemini 3.8 Flash	`high`	65.5	6.15	—	—	—	—
Qwen 3.8 27B	`xhigh`	29.2	0.00	—	—	—	—
GPT-5.6 Luna	`high`	38.1	0.11	50.4	0.04	—	—
	`xhigh`	56.6	0.21	56.6	0.06	—	—
	`max`	60.2	0.40	55.8	0.10	—	—

E Complete reasoning-effort sweep results

Table 6 lists all single-model and AgentBaton configurations plotted in Figure 5, including dominated configurations. Qwen and Luna denote Qwen 3.8 27B and GPT-5.6 Luna, respectively. The Qwen baseline uses a 1M-token context window.