ASKING YOUR ADVISOR: LEARNING WHEN TO CONSULT STRONGER MODELS IN AGENTIC SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

To balance task performance and computational cost, agentic workflows can use a lower-cost model for execution and selectively consult a stronger model. In such executor–advisor systems, the executor decides when to seek advice, making escalation depend on self-assessment, although current language models struggle to detect where their own reasoning fails. The need for stronger reasoning emerges during execution and varies across workflow phases. We introduce ARIA, a meta-controller that learns when to consult by balancing task success against consultation cost. Our approach formulates consultation as a sequential resource-allocation problem. At workflow contact points, ARIA maps execution evidence to a compact state abstraction and selects SILENT or CONSULT, with action values learned by Monte Carlo control from cost-penalized outcomes. Across evaluated tasks of SWE-bench Verified, Terminal-Bench 2.1, and τ^2-bench, ARIA recovers 44–81% of the executor-advisor success gap at less than half the advisor model's cost. ARIA achieves the best observed success-cost trade-off among ten other evaluated conditions, including routing and cascading methods, fixed and prompted consultation, and random placement at matched advice volume. The policies are compact and inspectable, with a single consultation-price parameter governing the success–cost trade-off without retraining either underlying model.

1 INTRODUCTION

Agentic systems increasingly tackle complex tasks such as software engineering (Chowdhury et al., 2024), data science (Merrill et al., 2026), and customer support (Barres et al., 2025) through long-running workflows that extend beyond isolated queries (OpenAI, 2025; Anthropic, 2024; Mohan & Kilpatrick, 2026). Sustaining these workflows over long task horizons is costly, as each stage may require reasoning, tool use, and revision. However, prior work shows substantial scope for reducing inference cost while maintaining similar task performance (Kapoor et al., 2025), and the need for stronger reasoning varies throughout execution. The challenge is therefore to allocate capability selectively, using stronger reasoning where its expected benefit justifies its additional cost.

Existing capability-allocation methods are selective, but mostly operate at the model or task level, making allocation decisions from the input before execution. Specialization *changes* the executor (Wang et al., 2024; Belcak et al., 2025), routing determines *which model* acts (Aggarwal et al., 2024; Ong et al., 2025), and adaptive computation controls *how much reasoning* it receives before execution (Snell et al., 2025; Damani et al., 2025). These approaches reduce reliance on costly frontier-model capability, but do not allocate stronger reasoning to specific moments within an evolving workflow. In long-running execution, the value of stronger reasoning can change as new observations, errors, or recovery opportunities emerge. A routine task may contain only a few decision points where additional capability is particularly valuable.

Executor-advisor systems provide a natural mechanism for introducing stronger reasoning into a workflow. A lower-cost model remains responsible for execution, while a stronger advisor provides guidance without taking control (Anthropic, 2026). This allows routine steps to remain with the executor and advisor input to be added selectively as the workflow unfolds. The remaining question is *when* to consult. Current approaches leave this to the executor itself, making escalation depend on self-assessment. Previous work has shown that large language models (LLMs) struggle to localize their own errors and have poorly calibrated confidence (Tyen et al., 2024; Xiong et al., 2024). Uncer-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

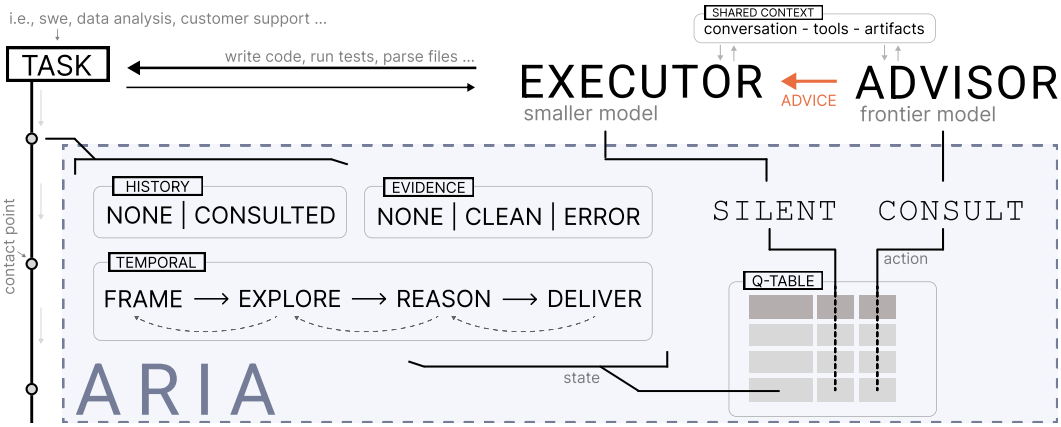

Figure 1: ARIA setup. At each contact point, the meta controller chooses between SILENT and CONSULT. Consultation adds guidance from a stronger advisor to the shared context.

tainty alone does not indicate whether consultation is useful. An executor may recover, while advice may be valuable even when it appears confident. The relevant question is whether consulting at this point improves the final outcome enough to justify its cost. This separates consultation allocation from a complementary problem: whether the executor incorporates the advice it receives.

We introduce **ARIA** (Allocation of Reasoning In Agentic systems), a learned meta-controller for selective expert consultation in agentic workflows. ARIA regulates access to a stronger advisor while keeping a lower-cost executor in control. We instantiate ARIA on Magentic-One (Fourney et al., 2024), while the controller itself only assumes recurring decision points: At workflow decision points, such as replanning or final delivery, it chooses whether the executor continues or receives guidance. We formulate this policy as a sequential decision problem and learn it with tabular Monte Carlo control. The state is derived from observable workflow evidence rather than self-reported confidence, and neither language model is fine-tuned. This way, ARIA can learn an optimal trade-off between two deployment extremes by allocating advisor to selected points in the trajectory.

We evaluate ARIA across three agentic settings: SWE-bench Verified (Chowdhury et al., 2024), Terminal-Bench 2.1 (Merrill et al., 2026), and τ^2-bench (Barres et al., 2025), spanning homogeneous, heterogeneous, and interactive workflows across software engineering, data analysis, system administration and customer service. We anchor the comparison with *Executor-only* and *Advisor-only* baselines as the lower and stronger-model reference points, and compare fixed, prompted, learned and allocation methods from related work between these reference points. A *Random*-placement control approximates ARIA's consultation rate to separate the effect of consultation timing from simply having advisor access. We report task success with realized model cost and evaluate how much of the executor-advisor performance gap selective consultation can recover.

Our contributions are threefold. First, we introduce ARIA, a modular and interpretable framework for learning consultation policies over executor–advisor pairs. Second, we evaluate ARIA across three agentic domains, analyzing success, cost and efficiency. Third, we analyze failures in executor–advisor collaboration to gain further insights into inter-agent coordination. We release the framework and traces for research on capability allocation and agent collaboration.

2 Related Work

Specialized models and adaptive computation. Specialization and adaptive computation improve efficiency without conditioning on evidence revealed during the running workflow. Small language models can be trained or fine-tuned for recurring tasks (Belcak et al., 2025). *CodeAct* fine-tunes models on agent-environment interactions, while reasoning models specialize for mathematical and code reasoning through supervised fine-tuning and reinforcement learning (Wang et al., 2024; Liu et al., 2026). Inference-time methods instead allocate additional computation to improve reasoning (Snell et al., 2025). Input-adaptive allocation predicts the benefit of computation budgets from the query alone and assigns resources before generating a response (Damani et al., 2025). ARIA differs by allocating capability after execution has begun, using evidence revealed along the trajectory.

Routing, cascading, and orchestration. Routing and orchestration allocate capability dynamically, but typically do so by changing which model executes or how execution is structured. *RouteLLM* selects between stronger and weaker models (Ong et al., 2025), while *AutoMix* uses self-verification and a POMDP cascade to decide whether a smaller model's answer should be accepted or escalated (Aggarwal et al., 2024). Allocation can operate at finer granularity: STEER routes reasoning steps between smaller and larger models using logit-derived confidence estimates, requiring access to model-internal signals unavailable in our setup (Lee et al., 2026). At the system level, *AFlow* optimizes workflow structures, while *Automated Design of Agentic Systems* searches over agent architectures from execution feedback (Zhang et al., 2025; Hu et al., 2025). ARIA instead takes the executor and workflow as given and learns when to acquire guidance without model-internal signals.

External consultation. Existing approaches make advice available during execution, but typically leave the decision *when* to request it to the agent itself. Action-advising work treats guidance as a limited resource and shows that its effect can depend on when a fixed budget is spent (Torrey & Taylor, 2013). In these settings, however, advice typically serves to accelerate the learning of an agent's task policy. These concepts are also emerging in human-agent systems that provide several interaction patterns: *Magentic-UI* supports co-planning and human intervention during execution (Mozannar et al., 2025), while *Collaborative Gym* incorporates communication into the agent's action space and studies prompted situational planning (Shao et al., 2026). Related ideas have recently appeared in agent-agent collaboration. Anthropic's Advisor Strategy pairs a lower-cost executor with a stronger advisor that can be consulted during execution through a tool call (Anthropic, 2026). *SWE-Protégé* goes further by training a smaller software-engineering agent to recognize when expert help is needed and act on the resulting guidance (Kon et al., 2026), whereas ARIA keeps the models fixed. Our contribution is therefore not the introduction of advice itself. ARIA instead learns a cost-aware policy for when to consult the advisor during execution, without modifying the underlying model.

3 METHODOLOGY

ARIA keeps the executor, advisor, environment and workflow fixed and learns only a separate binary policy that explicitly determines when advisor guidance is allocated during execution.

3.1 CONSULTATION DURING THE TRAJECTORY

Agentic workflows often involve recurring points at which agents must decide whether to continue, replan, or terminate in response to execution feedback (Agashe et al., 2025; Huang et al., 2023). We introduce opportunities for consultation at these decision points. Using Magentic-One as our execution framework (Fourney et al., 2024), we define three orchestration events as consultation contact points: PRE-STEP, before the next executor instruction; STALL, when stalled progress triggers replanning; and PRE-DELIVERY, immediately before final delivery. The contact-point type determines why a consultation opportunity occurs, whereas the temporal state in Section 3.2 separately captures coarse progress over the sequence of contact points.

At each contact point t, ARIA selects $a_t \in \{\text{SILENT}, \text{CONSULT}\}$. Under SILENT, the executor remains in control. Under CONSULT, the advisor receives task specification and current trajectory information and returns a natural-language assessment with recommended next steps. The orchestrator incorporates this guidance into the executor's subsequent instruction, while the executor remains responsible for reasoning, tool use, and final submission. Consultation can affect later actions, observations, and contact points through the executor's response to advice; exact intervention semantics are given in Appendix A.1. ARIA learns consultation value from downstream outcomes rather than immediate local feedback.

3.2 STATE ABSTRACTION

ARIA conditions its decision on existing signals available from the workflow rather than on the executor's own assessment, motivated by known limitations of LLM self-diagnosis and confidence calibration (Tyen et al., 2024; Xiong et al., 2024). We summarize the recorded history H_t along three dimensions: temporal position p_t (4 categories), execution evidence e_t (3), and consultation history h_t (2),

$$s_t = \phi(H_t) = (p_t, e_t, h_t), \qquad |\mathcal{S}| = 4 \times 3 \times 2 = 24.$$

Each dimension distinguishes circumstances under which the value of consultation may differ.

The contextual dimension captures what execution has revealed. Execution feedback can support correction (Chen et al., 2024b; Shinn et al., 2023), while successful self-repair depends strongly on the feedback available (Olausson et al., 2024; Huang et al., 2024). We therefore use $e_t \in \{\text{NONE}, \text{CLEAN}, \text{ERROR}\}$ to distinguish absent evidence from output without or with a recognized error signal. This reflects observed execution status rather than correctness: clean output may follow a wrong strategy, and an error may be recoverable unaided. The temporal dimension captures where this evidence occurs and is derived independently of the contact-point type. PRE-DELIVERY maps directly to DELIVER; otherwise, the first contact point maps to FRAME, the next two to EXPLORE, and later contact points to REASON. Inspired by Pólya (1945), $p_t \in \{\text{FRAME}, \text{EXPLORE}, \text{REASON}, \text{DELIVER}\}$ provides a task-agnostic representation of coarse workflow progress. The consultation-history dimension, $h_t \in \{0, 1\}$, records whether advisor guidance has already entered the trajectory. Because prior consultation changes the information available to the executor, the value of a further request may differ, a distinction also reflected in contemporary executor–advisor systems (Anthropic, 2026). Remaining budget is omitted from s_t; once B is exhausted, CONSULT is forced to SILENT. Thus, the policy does not condition explicitly on remaining budget. The 24-state abstraction keeps the policy compact and inspectable by summarizing the trajectory into a small number of workflow signals; Appendix A.2 specifies the exact state mapping.

3.3 COST-AWARE POLICY LEARNING

The state representation provides no intermediate signal of consultation value. ARIA therefore learns from the terminal task outcome. For trajectory τ, let $Y(\tau) \in \{0, 1\}$ denote task success and $N_c(\tau)$ the number of consultations. We define the cost-aware return

$$G(\tau) = Y(\tau) - \lambda \frac{\bar{c}}{\Delta} N_c(\tau), \tag{1}$$

where $\bar{c}$ is the mean consultation cost and Δ the mean per-episode spend of consulting at every contact point, measured per domain (Appendix A.3). Thus, $\bar{c}/\Delta$ normalizes one consultation against the spend of consulting throughout, making λ approximately the price of that policy in units of one solved task. We assign a fixed penalty to each consultation, treating advisor access as a count-limited resource, consistent with prior work on advice budgets and constant intervention costs (Torrey & Taylor, 2013; Cortes et al., 2016). Maximizing the expected return in equation 1 yields a fixed-multiplier Lagrangian relaxation of maximizing expected task success subject to a consultation-cost budget (Altman, 1999), with $\kappa = \lambda\bar{c}/\Delta$ as the effective per-consultation penalty. LLM collaboration systems similarly use a fixed deferral penalty to control escalation behavior (Yang et al., 2026).

Because the task outcome arrives only at the end of the episode, ARIA learns action values using every-visit tabular Monte Carlo control with sample-average updates (Sutton & Barto, 2018). A visit at contact point t is assigned the remaining episodic return $G_t = Y(\tau) - \kappa\, N_c^{\geq t}(\tau)$, the task outcome minus penalties for consultations at or after t; the episodic return in equation 1 is G_0. We set $\gamma = 1$: discounting would reduce the terminal outcome relative to immediate consultation cost and bias the policy toward later consultation. Training uses an ϵ-greedy behavior policy, and only episodes reaching a terminal outcome update the table. Evaluation uses the frozen greedy policy, with ties, including unvisited states, resolving to SILENT. Further details are reported in Appendix A.3.

4 EXPERIMENTS

We evaluate ARIA across three agentic environments using fixed proprietary models. Direct baselines are limited to methods that do not require model internals or fine-tuning. We test whether learned consultation improves the success–cost trade-off over autonomous and task-level baselines, whether placement matters beyond advisor access, how consultation shifts the operating point, and what the results reveal about agent-agent collaboration.

Benchmarks and models. We evaluate software engineering on SWE-bench Verified (Chowdhury et al., 2024), heterogeneous terminal workflows on Terminal-Bench 2.1 (Merrill et al.,

2026), and interactive customer service on τ^2-bench (Barres et al., 2025). Train and test tasks are disjoint. We use 50/50 train/test tasks for SWE-bench, with Verified-Mini as the test set, 27/50 for Terminal-Bench and 20/40 tasks for τ^2-bench. Appendix B.1 details all benchmark splits. The executor is `gemini-3-flash-preview` at low reasoning effort and the advisor is `gemini-3.1-pro-preview` at high reasoning effort. Using models from the same family reduces cross-model differences in deployment behavior. Consultation conditions use at most $B = 4$ advisor calls per episode; budget sensitivity is examined in Section 5.1.

Evaluation conditions. Our baselines are chosen to test competing explanations for the benefit of learned consultation. *Executor-only* and *Advisor-only* establish the capability range. *FrugalGPT* (Chen et al., 2024a), *RouteLLM* (Ong et al., 2025), and *AutoMix* (Aggarwal et al., 2024) test whether routing or cascading provides sufficient allocation against ARIA under the same model pair. *Plan*, motivated by Magentic-UI's upfront workflow planning (Mozannar et al., 2025), *Heuristic*, derived from Anthropic's recommended start/end and stall triggers (Anthropic, 2026), and *Always* test whether defined consultation schedules or advisor access without learned timing are sufficient. *LLM-Router* tests whether the native LLM orchestrator can decide when to involve the advisor without a learned controller, while *Random* separates informed timing from consultation volume. Together, these baselines isolate the roles of model strength, allocation strategy, advisor access, and consultation timing. Appendices B.2 and B.3 provide baseline details and *Random*-control calibration.

Table 1: Baseline design for selective consultation, covering capability references, task-level allocation, fixed schedules, LLM-based timing, uninformed placement, and learned allocation.

Condition	Allocation rule	Tests for
Executor-only	no advisor	lower-cost reference
Advisor-only	advisor executes	stronger-model reference
FrugalGPT (Chen et al., 2024a)	score-based cascade	cost-aware task-level cascading
RouteLLM (Ong et al., 2025)	pre-task routing	learned task-level routing
AutoMix (Aggarwal et al., 2024)	verify then escalate	adaptive task-level escalation
Plan (Mozannar et al., 2025)	first contact	early intervention
Heuristic (Anthropic, 2026)	beginning/end triggers	vendor-reported heuristic
Always	at every contact point	advisor access without selection
LLM-Router	orchestrator agent selects advisor	LLM-decided/prompted consultation
Random	rate-matched advice	matched consultation volume
ARIA	state-conditioned policy	learned consultation timing

Consultation-price ablation. The default policy ($\lambda = 0.10$) is trained online under its own ϵ-greedy behavior policy. Policies for $\lambda \in \{0.35, 0.50\}$ are re-estimated offline from the same training trajectories by recomputing returns and the value table. Because the trajectories were collected under the $\lambda = 0.10$ behavior policy, this sweep is an offline policy re-estimation under that visitation distribution rather than equivalent retraining at each price (Appendix A.3).

Quantitative evaluation. Following *RouteLLM* (Ong et al., 2025), we report *performance gap recovered* (PGR), which measures the fraction of the executor–advisor success gap recovered by a condition. Because the executor remains the base model across evaluated baselines, consultation adds advisor computation to the *Executor-only* costs. We therefore normalize realized evaluation cost between the *Executor-only* (0) and *Advisor-only* (1) anchors:

$$\mathrm{PGR} = \frac{S - S_{\mathrm{exec}}}{S_{\mathrm{front}} - S_{\mathrm{exec}}}, \qquad \mathrm{CostShare} = \frac{C - C_{\mathrm{exec}}}{C_{\mathrm{front}} - C_{\mathrm{exec}}}.$$

PGR is normalized by the same anchors, such that *Executor-only* maps to 0 and *Advisor-only* to 1 on both axes. A task-independent mixture lies on $\mathrm{PGR} = \mathrm{CostShare}$ and thus has $\mathrm{ER} = 0$; positive values indicate a better success-cost trade-off than simple model mixing. We define *excess recovery* as $\mathrm{ER} = \mathrm{PGR} - \mathrm{CostShare}$, the vertical distance from this reference line. Positive ER therefore means that a method recovers more performance than its normalized cost share, with larger values indicating a better success-cost trade-off. Detailed token and cost reconciliation are reported in Appendix B.4.

Qualitative failure analysis. To analyze which failures remain beyond success rates and costs, we apply the Multi-Agent System Failure Taxonomy (MAST) (Cemri et al., 2025), which groups

14 failure modes into system-design (FC1), inter-agent misalignment (FC2), and task-verification (FC3). Each failed trajectory receives one primary failure mode and additionally, all modes that occur. MAST annotations are produced by an LLM judge and are therefore validated on a stratified subset against human annotations; the full setup and protocols are documented in Appendix C.6.

5 RESULTS

On SWE-bench, performance ranges from 18.0% at \$11.48 for *Executor-only* to 62.0% at \$45.90 for *Advisor-only*, with ARIA reaching 48.0% at \$22.09 (Table 2). The same pattern holds across the other benchmarks: ARIA achieves a performance gap recovered (PGR) of 68.2%, 44.4%, and 80.6% on SWE-bench, Terminal Bench, and τ^2-bench while using 48.1%, 40.9%, and 34.5% of *Advisor-only* evaluation cost. Its excess recovery ($\mathrm{ER} = \mathrm{PGR} - \mathrm{CostShare}$), measuring how much more performance gap is recovered than normalized cost is spent, is 0.37, 0.15, and 0.65, respectively, exceeding all other evaluated conditions. Moreover, no evaluated condition achieves higher mean success than ARIA at equal or lower cost, or lower cost at equal or higher mean success. Thus, ARIA lies on the empirical success–cost Pareto frontier among the evaluated conditions.

Table 2: Success-cost trade-offs over three seeds (standard deviations are reported in Appendix C.1)

	SWE-bench Verified			**Terminal-Bench 2.1**			**τ^2-bench Telecom**		
Condition	Succ. %	Cost \$	ER	Succ. %	Cost \$	ER	Succ. %	Cost \$	ER
Executor-only	18.0	11.48	0.00	21.3	9.76	0.00	72.5	5.07	0.00
Advisor-only	62.0	45.90	0.00	51.3	60.99	0.00	98.3	22.50	0.00
FrugalGPT	31.3	22.61	-0.02	22.0	11.10	0.00	78.3	7.02	0.11
RouteLLM	30.0	25.03	-0.12	28.0	21.66	-0.01	88.3	9.61	0.35
AutoMix	**52.0**	59.68	-0.63	22.0	16.51	-0.11	**97.5**	24.98	-0.17
Plan	24.0	10.90	0.15	24.0	11.34	0.06	70.0	5.69	-0.13
Heuristic	41.3	21.50	0.24	**34.7**	27.47	0.10	83.3	7.13	0.30
Always	34.0	21.39	0.08	**35.3**	27.94	0.11	87.5	9.01	0.36
LLM-Router	29.3	15.77	0.13	30.0	18.47	0.12	87.5	6.16	0.52
Random	28.0	17.33	0.06	30.7	22.25	0.07	85.8	8.24	0.33
ARIA	48.0	22.09	**0.37**	**34.7**	24.93	**0.15**	93.3	7.77	**0.65**

Learned consultation differs from task-level allocation. The task-level allocation baselines do not reproduce the same success-cost position as our approach. On SWE-bench, *FrugalGPT*, *RouteLLM*, and *AutoMix* achieve ER values of -0.02, -0.12, and -0.63, compared with 0.37 for ARIA. *RouteLLM* and ARIA also spend similar amounts (\$25.03 vs. \$22.09), yet solve 30.0% and 48.0% of tasks. The pattern is similar on Terminal-Bench, where all three task-level methods remain at or below the executor–advisor reference line while ARIA reaches $\mathrm{ER} = 0.15$. On τ^2-bench, *FrugalGPT* and *RouteLLM* improve over the reference line but remain below ARIA's ER of 0.65; *AutoMix* reaches 97.5% at an evaluation cost of \$24.98, 3.2× that of ARIA and higher than even *Advisor-only* execution. The evaluated allocation methods do not match ARIA's observed success-cost trade-off, suggesting value in learning allocation on evidence emerging during execution. *FrugalGPT* and *RouteLLM* use task-level information, while *AutoMix* additionally uses post-generation self-verification. None of these approaches incorporates evolving evidence throughout the agentic trajectory, specifically on the tested benchmarks.

Advisor access alone does not explain the result. The *Always* condition consults at every opportunity under the same budget as ARIA. On SWE-bench it costs nearly the same (\$21.39 vs. \$22.09) but achieves 14.0 pp less; on τ^2-bench it is both more expensive and 5.8 points less successful. Terminal-Bench provides the weaker contrast as *Always* gains 0.6 points in success but costs 12% more.

Table 3: ARIA relative to the autonomous anchors.

	SWE	**Terminal**	τ^2
PGR	68.2%	44.4%	80.6%
Cost / Pro	48.1%	40.9%	34.5%
Cost Share	30.8%	29.6%	15.5%
Sav. vs. equal-succ. mix	\$12.86	\$7.59	\$11.36

PGR follows the weak-strong performance-gap normalization of Ong et al. (2025). ER is defined as $\mathrm{PGR} - \mathrm{CostShare}$.

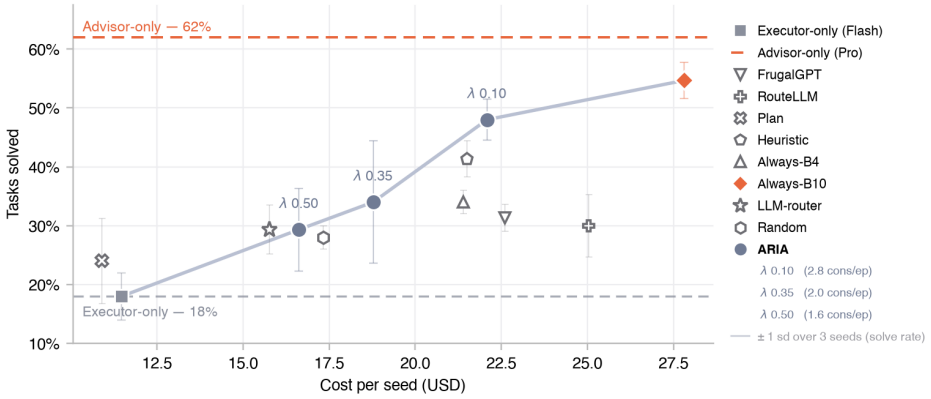

Figure 2: Success-cost operating points on SWE-bench for ARIA policies with $\lambda \in \{0.1, 0.35, 0.5\}$, consultation-budget ablations, and reference conditions.

5.1 Placement matters beyond advisor access and volume

Similar advice volume produces different outcomes. The aggregate comparisons do not yet separate consultation placement from consultation volume. Similar advice volume produces different outcomes: On SWE-bench, *Random*, *Heuristic*, and ARIA use 2.53, 2.75, and 2.78 consultations per episode, yet achieve 28.0%, 41.3%, and 48.0% success, respectively (Appendix C.2). ARIA gains 20.0 pp over uninformed placement at similar advice volume, with a smaller 6.7-point advantage over *Heuristic*. The effect is weaker on Terminal-Bench. *Random* reaches 30.7% success with 2.42 consultations per episode, while *Heuristic* and ARIA both reach 34.7% with 2.25 and 2.46 consultations; ARIA costs 9.2% less than *Heuristic*. On τ^2-bench, ARIA reaches 93.3% success versus 85.8% for *Random* and is slightly cheaper, but *Random* realizes only 87.8% of ARIA's consultation volume. The placement control is tightest on SWE-bench and weaker on τ^2.

More advice is not uniformly better. In the headline comparison, *Always* uses the same $B = 4$ consultation budget as ARIA in order to have equal conditions. Even under this matched budget, it falls short of ARIA by 14.0 points on SWE-bench and 5.8 points on τ^2-bench; on Terminal-Bench it gains 0.6 points at 12% higher cost. As a separate budget-sensitivity analysis, we increase the *Always* budget to $B = 10$ (Table 4). On SWE-bench, more advice raises success by 20.7 points and brings ER close to ARIA's, but at higher cost and nearly twice its consultation volume; Terminal-Bench gains only 4.0 points with unchanged ER, while on τ^2-bench success falls and ER turns negative.

Table 4: Change from *Always*-$B4$ to $B10$.

	ΔS	ΔC	ΔN_c	ΔER
SWE	+20.7	+$6.42	+1.95	+0.28
TB	+4.0	+$6.74	+0.82	0.00
τ^2	-2.5	+$5.04	+5.78	-0.38

Native orchestrator routing is sparse and late. *LLM-Router* removes ARIA's learned policy and lets the native orchestrator agent select the advisor through speaker selection. Despite repeated opportunities to consult, each informed by the full task ledger, the orchestrator requests an average of only 0.41, 0.61, and 0.48 consultations per episode on SWE-bench, Terminal-Bench, and τ^2-bench. Its requests often occur in similar workflow phases as ARIA but substantially later: 77%, 64%, and 76% of mapped *LLM-Router* consultations occur in the final third, compared with 47%, 25%, and 7% for ARIA. Native routing therefore identifies similar phases but not ARIA's timing or consultation frequency.

5.2 Consultation price controls the operating point

The consultation price sets ARIA's operating point. Table 2 reports the $\lambda = 0.10$ policy, which attains the highest excess recovery among the evaluated conditions on SWE-bench (ER $= 0.37$). Figure 2 shows how this point changes as consultation price increases. Increasing λ from 0.10 to 0.35 and 0.50 reduces consultation from 2.78 to 2.03 and 1.55 calls per episode and evaluation costs from \$22.09 to \$18.78 and \$16.63, while success falls from 48.0% to 34.0% and 29.3%.

The normalized success-cost view shows that the cheaper policies are not more efficient. ER falls from 0.37 at $\lambda = 0.10$ to 0.15 and 0.11 at the higher prices. Thus, $\lambda = 0.10$ is the highest-ER

operating point among the tested values: the additional consultations recover more performance than the corresponding increase in normalized cost. *Always-$B10$* reaches nearly the same efficiency on SWE-bench (ER $= 0.36$) only at roughly twice ARIA's consultation volume (Section 5.1). The three ARIA policies are re-estimated from the same training traces (Section 4), so this price comparison characterizes offline sensitivity to consultation price under the observed visitation distribution rather than independent retraining at each price. As the price increases, ARIA consults less often and changes where consultation remains worthwhile. Section 5.3 examines this structure directly.

5.3 Learned allocation differs across workflow domains

The consultation-price sweep shows how much advice ARIA retains, while the frozen policy tables reveal where it allocates that advice (Appendix C.3). A clear pattern emerges: consultation is concentrated in a small subset of execution states. Each learned policy selects CONSULT in only 4–9 of the 24 nominal states, so most contact points are left silent by decision rather than by lack of experience. On τ^2-bench, the states not encountered during training are exclusively error-evidence states. Unlike SWE-bench and Terminal-Bench, where coding and terminal actions can expose explicit failures, τ^2 primarily involves interactive dialogue and tool use, so mistakes often remain semantic or procedural rather than producing hard error signals. Its comparatively high baseline success further means that such explicit failure states are rarely encountered. Consultation is thus selective and cannot be explained solely as a response to explicit errors.

Beyond this shared pattern, allocation differs by benchmark: 53.5% of realized advisor calls on SWE-bench occur at the last temporal stage, DELIVER, whereas 82.2% on τ^2-bench occur during REASON. By contrast, the Anthropic-inspired *Heuristic* condition triggers consultation before committing to an approach and before declaring completion, with additional calls when progress stalls or the approach changes (Anthropic, 2026). These generic triggers do not consistently align with ARIA's learned consultation patterns. On SWE-bench, our approach places substantially more of its realized consultation at the delivery stage; on τ^2-bench, consultation instead concentrates in the middle of the trajectory. Correspondingly, the *Heuristic* trails ARIA by 6.7 pp on SWE-bench and 10.0 points on τ^2-bench, while tying it on the more heterogeneous Terminal-Bench. Together, these findings suggest that generic consultation rules provide a useful starting point, but benchmark-specific allocation can offer further improvement. Cross-seed policy agreement and a complementary within-state analysis of CONSULT versus SILENT are reported in Appendices C.4 and C.5.

5.4 Failure analysis reveals mechanism and scope

To understand how consultation changes the failures that remain, we classify failed trajectories with MAST (Cemri et al., 2025) and focus here on their primary failure mode (Figure 3); secondary failure modes are reported in Appendix C.6. On SWE-bench, autonomous execution is dominated by task-verification failures (*Executor-only*: 86%). As consultation placement becomes more targeted, verification-primary failures account for a smaller share of residual failures: 68% under *Random*, 46% under *Heuristic*, and 34% under ARIA. *LLM-Router* remains close to the autonomous profile despite access to the same advisor (83%). In absolute terms, 26 verification-primary failures remain under ARIA, compared with 70 under *Random*. Consultation therefore changes not only how often the system fails, but also what remains unresolved.

As verification decreases, a different bottleneck emerges. Ignored agent input (FM-2.5) becomes ARIA's dominant residual mode, accounting for 58% of its SWE-bench failures. The mode is absent under the orchestrator agent routing (*LLM-Router*: 0%), less common under uninformed placement (*Random*: 16%), and increasingly prominent under externally allocated advice (*Heuristic*: 47%; *Always-$B10$*: 71%). Across our experiments, FM-2.5 is substantially more prevalent than in MAST-Data (17.3% vs. 1.9%) (Cemri et al., 2025). Human validation confirms that these cases contain relevant advice that reached the executor but was not acted upon (Appendix C.6). Placement and uptake are therefore distinct problems: ARIA can learn where advice has high downstream value without ensuring that the executor follows every recommendation. This is evident on SWE-bench at the DELIVER stage, where ARIA often recommends consultation despite the highest ignored-input rate across stages (36–40%). On τ^2-bench, the ignored-input rate remains below 7% throughout.

The same distinction helps explain the cross-benchmark results. On Terminal-Bench, ARIA and *Heuristic* achieve the same success rate, but each solves 12 episodes the other fails. Their remain-

ing failures also differ: verification-primary failures account for 53% under ARIA, compared with 82% under *Heuristic*. Thus, different consultation schedules can produce the same headline performance while succeeding on different tasks and exposing different residual limitations. On τ^2-bench, in contrast, ARIA leaves only eight judged failures, so the SWE-style uptake bottleneck does not emerge. More generally, MAST profiles depend on the interaction regime as well as on the participating models: communication can remove some failure modes while making others possible. Failure taxonomies should therefore be interpreted as properties of the complete multi-agent system, not of its component models in isolation.

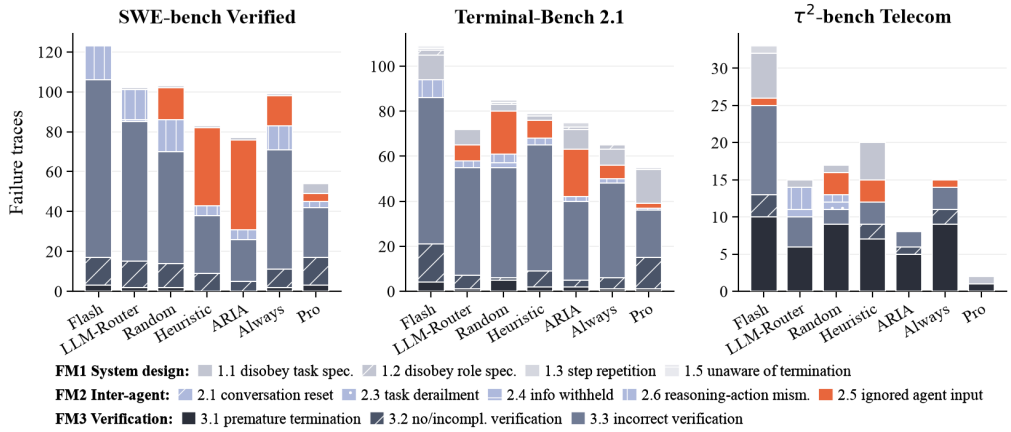

Figure 3: Primary failure mode across baselines and benchmarks grouped into system-design, inter-agent misalignment, and verification failures, showing different residual failure profiles.

6 Discussion and Limitations

Our results show that consultation timing matters more than access or volume alone. ARIA achieves the highest excess recovery on all three benchmarks, outperforming both fixed allocation and self-assessment. Gains are largest when similar decision points repeat across tasks and smaller on heterogeneous tasks such as on Terminal-Bench, suggesting that consultation policies are most useful when learned within related workflows rather than across widely different task types. Although our experiments focus on a single model pair and the Magentic-One framework, ARIA provides a testbed for studying other model pairs, agent configurations, and orchestration frameworks.

Beyond these empirical findings, ARIA makes the learned division of labor inspectable rather than leaving consultation decisions implicit in the underlying models. Its compact policy requires neither fine-tuning nor access to model internals and separates two decisions often conflated in multi-agent systems: when to consult an advisor and what happens once that guidance is provided. Our failure analysis shows that improving the first can expose advice uptake as the next bottleneck, making ARIA both a resource-allocation mechanism and a controlled way to study inter-agent coordination.

7 Conclusion

ARIA learns when an executor should consult an advisor during agentic workflows. Across three benchmarks, it recovers 44–81% of the executor-advisor performance gap at less than half the *Advisor-only* cost and achieves the highest empirical excess recovery among all evaluated conditions. It requires neither fine-tuning nor access to model internals, and represents its decisions as an inspectable policy with a single parameter controlling the consultation trade-off. Together, these results establish selective consultation as an effective mechanism for executor-advisor collaboration, while the failure analysis identifies advice uptake as a further lever for improving efficiency.

AI USE STATEMENT

Generative AI tools, including ChatGPT (Astra) and Claude (Fable 5, Opus 5), were used during the preparation of this work to support revision, literature discovery and organization, research ideation, methodological cross-checking, technical implementation and analysis support, figure preparation, and the drafting of selected manuscript sections. AI-assisted outputs were treated as suggestions rather than authoritative sources and were reviewed, edited, and verified by the authors before inclusion. All citations, quantitative results, experimental claims, benchmark splits, tables, figures, and methodological descriptions were checked against the underlying literature, implementation, experiment logs, or source artifacts. The authors take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

The consultation objective, state construction, Monte Carlo updates, exploration schedule, training budgets, and frozen deployment policy are specified in Section 3 and Appendix A. Appendix B documents the benchmark splits, baseline implementations, *Random*-control calibration, model settings, and token- and cost-accounting procedure. Appendix C reports seed-level variation, controller diagnostics, policy agreement, within-state analyses, and the MAST failure-analysis protocol. All headline results are averaged over three evaluation seeds, with corresponding dispersion reported separately. An anonymized repository provides the implementation and the code.[1] The supplementary material submitted through OpenReview contains the MAST failure-analysis materials, result data supporting the reported analyses, and a subset of agent traces. The complete trace release will be made available upon acceptance.

REFERENCES

David Abel, D. Ellis Hershkowitz, and Michael L. Littman. Near optimal behavior via approximate state abstraction. In *Proceedings of the International Conference on Machine Learning*, volume 48, pp. 2915–2923. PMLR, 2016.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent S: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/394c7c30ea87b5c3521b4d9e9d419071-Abstract-Conference.html.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4164.

Eitan Altman. *Constrained Markov decision processes*. Routledge, 1 edition, 1999. doi: 10.1201/9781315140223.

Anthropic. Building effective agents. Anthropic Engineering, December 2024. URL https://www.anthropic.com/engineering/building-effective-agents.

Anthropic. The advisor strategy: Give agents an intelligence boost. Claude Blog, April 2026. URL https://claude.com/blog/the-advisor-strategy.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment, 2025.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small language models are the future of agentic AI, 2025.

[1] https://anonymous.4open.science/r/aria-DB58

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4082.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=cSimKw5p6R.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=KuPixIqPiq.

Neil Chowdhury, James Aung, Jun Shern Chan, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. Learning with rejection. In *Proceedings of the International Conference on Algorithmic Learning Theory*, pp. 67–82. Springer, 2016. doi: 10.1007/978-3-319-46379-7_5.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/ff414825df833edb8b1839e3d5d495e9-Abstract-Conference.html.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the Conference on Robot Learning*, volume 205, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Trans. on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=Zy4uFzMviZ.

Patrick Tser Jern Kon, Archana Pradeep, Ang Chen, Alexander P. Ellis, Warren Hunt, Zijian Wang, John Yang, and Samuel Thompson. SWE-Protégé: Learning to selectively collaborate with an expert unlocks small language models as software engineering agents, 2026.

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*, pp. 31483–31491, 2026. doi: 10.1609/aaai.v40i37.40413.

Zihan Liu, Zhuolin Yang, Yang Chen, Chankyu Lee, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. AceReason-Nemotron 1.1: Advancing math and code reasoning through SFT and RL synergy. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/72398f80dc0f81159f79e6ff8fd1e376-Abstract-Conference.html.

Mike A. Merrill et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/444a3737adaee10d86ad2ef5f74468e6-Paper-Conference.pdf.

Varun Mohan and Logan Kilpatrick. Building the agentic future: Developer highlights from I/O 2026. Google, The Keyword, May 2026. URL https://blog.google/innovation-and-ai/technology/developers-tools/google-io-2026-developer-highlights/.

Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, Eric Zhu, Griffin Bassman, Jacob Alber, Peter Chang, Ricky Loynd, Friederike Niedtner, Ece Kamar, Maya Murad, Rafah Hosn, and Saleema Amershi. Magentic-UI: Towards human-in-the-loop agentic systems, 2025.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=y0GJXRungR.

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 from preference data. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5503a7c69d48a2f86fc00b3dc09de686-Paper-Conference.pdf.

OpenAI. New tools for building agents. OpenAI, March 2025. URL https://openai.com/index/new-tools-for-building-agents/.

George Pólya. *How to solve it: A new aspect of mathematical method*. Princeton University Press, 1945.

Yijia Shao, Vinay Samuel, Yucheng Jiang, John Yang, and Diyi Yang. Collaborative Gym: A framework for enabling and evaluating human-agent collaboration. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/a1c8a68e52499c9396854e3f967e37c0-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0377.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, second edition, 2018.

Lisa Torrey and Matthew E. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1053–1060, 2013.

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL*, pp. 13894–13908. Association for Computational Linguistics, August 2024. doi: 10.18653/v1/2024.findings-acl.826.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the International Conference on Machine Learning*, volume 235, pp. 50208–50232. PMLR, 2024. URL https://proceedings.mlr.press/v235/wang24h.html.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? An empirical evaluation of confidence elicitation in LLMs. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6733cf15e10e2cd1d59af033c3bb8507-Abstract-Conference.html.

Wei Yang, Defu Cao, Jiacheng Pang, Muyan Weng, and Yan Liu. Adaptive collaboration with humans: Metacognitive policy optimization for multi-agent LLMs with continual learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=IKVUB9Exuc.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

A METHODOLOGICAL DETAILS

A.1 ADVISOR INTERVENTION SEMANTICS

Section 3.1 defines consultation as a binary decision at recurring workflow contact points. Here, we specify the intervention contract used once consultation is triggered. Across ARIA, *Random*, *Heuristic*, *Always*, and *Plan*, this contract is held fixed; the conditions differ only in the rule that determines which eligible contact points trigger consultation.

When a condition selects CONSULT, the advisor has access to the task specifications and the full available trajectory context without additional summarization or truncation, and returns natural-language guidance with recommended next steps. The advisor is read-only: it does not execute tools, take environment actions, or submit the final answer. The executor remains responsible for all subsequent reasoning, tool use, environment interaction, and final submission, and may follow, adapt, or ignore the recommendation. Consultation therefore provides additional reasoning without transferring execution control to the advisor.

Contact-point-specific intervention. How the guidance enters the trajectory is determined solely by the contact-point type defined in Section 3.1. At `PRE_STEP`, the advisor response is appended to the orchestrator's next instruction before execution continues. At `STALL`, guidance is inserted before replanning so that the revised plan can incorporate the advisor's assessment. At `PRE_DELIVERY`, the advisor reviews the current solution before final submission; if it flags the trajectory as off track, the workflow permits one additional revision cycle before finalization. These insertion rules are identical across the consultation conditions above.

Silent action and downstream effects. Under SILENT, no advisor call is made and the underlying workflow continues unchanged. Thus, the meta-controller's direct intervention is limited to whether advisor guidance enters the trajectory; the executor, advisor, task interface, and surrounding workflow otherwise remain fixed. Once introduced, however, advice can indirectly change later executor actions, observations, and contact points through the executor's response to that guidance. The *LLM-Router* baseline follows a separate routing contract, described in Appendix B.2. This separation allows consultation allocation and advice uptake to be analyzed independently.

A.2 STATE CONSTRUCTION

Section 3.2 defines the compact state abstraction $s_t = (p_t, e_t, h_t)$. Here, we specify the exact mapping from workflow events and observable execution feedback to each state component.

Design principle. ARIA maps the observable workflow history H_t to the compact state $s_t = (p_t, e_t, h_t)$ defined in Section 3.2. The representation is intentionally based on workflow-level signals rather than model-internal confidence or self-assessment, whose reliability is known to be limited (Tyen et al., 2024; Xiong et al., 2024). The abstraction is designed to pool consultation experience across recurring execution contexts rather than reconstruct the full trajectory. This follows the general motivation of approximate state abstraction: a smaller representation improves reuse and interpretability at the cost of discarding potentially decision-relevant information (Abel et al., 2016).

Temporal-state mapping. Magentic-One provides recurring orchestration events during execution (Fourney et al., 2024). We use three such events as consultation contact points: `PRE_STEP`, before the next executor instruction; `STALL`, when stalled progress triggers replanning; and `PRE_DELIVERY`, immediately before final delivery. Contact-point type and temporal state are distinct: the former identifies why a consultation opportunity occurs, whereas p_t provides a coarse representation of where that opportunity lies in the trajectory.

In the experimental setup used throughout this paper, the temporal state is assigned as

$$p_t = \begin{cases} \textsc{Frame}, & \text{if } \mathrm{kind}_t \neq \texttt{PRE_DELIVERY} \text{ and } i_t = 0, \\ \textsc{Explore}, & \text{if } \mathrm{kind}_t \neq \texttt{PRE_DELIVERY} \text{ and } i_t \in \{1, 2\}, \\ \textsc{Reason}, & \text{if } \mathrm{kind}_t \neq \texttt{PRE_DELIVERY} \text{ and } i_t \geq 3, \\ \textsc{Deliver}, & \text{if } \mathrm{kind}_t = \texttt{PRE_DELIVERY}, \end{cases}$$

where i_t is the zero-based index over all contact points in the episode. All reported experiments use this fixed mapping. Although the concrete mapping is specific to Magentic-One, the abstraction is designed to generalize: other single- or multi-agent systems can instantiate the same idea using their own recurring workflow events and corresponding progress signals. The controller records the index before constructing the state and increments it afterwards. Consequently, `STALL` creates an additional decision opportunity but does not define a temporal stage of its own. Its temporal label is determined by the same ordinal rule as other non-delivery contact points. The labels are inspired by coarse stages of problem solving (Pólya, 1945); they should be interpreted as task-agnostic progress labels rather than semantic classifications of the executor's reasoning.

Execution-evidence mapping. The evidence dimension summarizes observable execution feedback. Such feedback can support correction and self-repair (Chen et al., 2024b; Shinn et al., 2023), although its usefulness depends on which signals are exposed by the environment (Olausson et al., 2024; Huang et al., 2024). Accordingly, e_t records execution status rather than semantic correctness.

For SWE-bench and Terminal-Bench, the implementation considers the most recent step containing executed output. If no such step exists, $e_t = \text{NONE}$. Otherwise, the step maps to ERROR when its execution-error field is set and to CLEAN otherwise. The error field is derived from observable execution output using explicit failure signals, including tracebacks or exceptions, non-zero exit codes, explicit failure markers, and non-zero pytest failure summaries. The most recent step determines the state, so a failure followed by a successful repair does not remain encoded as ERROR.

For τ^2-bench, the interaction environment exposes tool results rather than conventional process output. A step is therefore treated as acting when it produces either executed output or a harness-recorded tool result. If no acting step exists, $e_t = \text{NONE}$; explicit machine- or tool-reported errors map to the failure state, and other tool results map to the clean state. The implementation names these two non-empty values OK and FAILED; throughout the paper they correspond to CLEAN and ERROR, respectively. End-of-conversation notifications are not encoded as execution failures.

Consultation-history mapping. The final state component, $h_t \in \{0, 1\}$, records whether advisor guidance has already entered the trajectory. This distinction allows the controller to represent that the value of a further consultation may differ once the executor has already received advice. Remaining consultation budget is deliberately omitted from the state; once the episode-level budget B is exhausted, subsequent CONSULT choices are forced to SILENT. Together, these mappings produce the 24 state representation used by the controller. The abstraction preserves three factors that can systematically change consultation value: trajectory progress, observable execution evidence, and prior advisor access, while remaining small enough for direct tabular estimation and inspection.

Together, these deterministic mappings make every deployment state directly recoverable from observable workflow signals while keeping the meta-controller small enough for tabular estimation and inspection.

A.3 TRAINING AND POLICY ESTIMATION

ARIA learns only the consultation controller; the executor and advisor remain fixed throughout training. This section details how the cost-aware objective in Eq. 1 is used to estimate action values and derive the frozen deployment policy.

Training objective and cost calibration. Each consultation incurs the fixed penalty $\kappa = \lambda\bar{c}/\Delta$, where $\bar{c}$ is the mean consultation cost and Δ is the mean per-episode spend of consulting at every contact point. All headline policies use $\lambda = 0.10$. Table 5 reports the benchmark-specific calibration values and the resulting penalties. These penalties define the learning objective; realized evaluation costs are reported separately.

Table 5: Benchmark-specific cost calibration for the headline policies at $\lambda = 0.10$. The quantities $\bar{c}$ and Δ are in USD; $\kappa = \lambda\bar{c}/\Delta$ is the per-consultation penalty in reward units.

Benchmark	$\bar{c}$	Δ	κ
SWE-bench	0.0490	0.1663	0.029465
Terminal-Bench	0.0683	0.2816	0.024254
τ^2-bench	0.0244	0.1281	0.019048

Action-value estimation. We use every-visit tabular Monte Carlo control with sample-average updates and $\gamma = 1$. At contact point t in trajectory τ, the update target is the return-to-go

$$G_t = Y(\tau) - \kappa N_c^{\geq t}(\tau),$$

where $Y(\tau)$ is terminal task success and $N_c^{\geq t}(\tau)$ counts consultations at or after that visit. Only episodes reaching a terminal task outcome contribute value updates. For each visited state-action pair (s, a),

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{j=1}^{N(s,a)} G_j, \tag{2}$$

where $N(s, a)$ is the number of visits to (s, a) and G_j is the return-to-go from the corresponding visit.

Every occurrence contributes one sample, retaining all observed state-action returns in the compact state space. Repeated visits within an episode therefore contribute correlated samples, and long trajectories can receive more weight than under first-visit Monte Carlo. The equivalent incremental learning rate is $\alpha = 1/N(s, a)$, so no additional learning-rate hyperparameter is introduced.

Exploration. Training follows an ϵ-greedy behavior policy over $\{\textsc{Silent}, \textsc{Consult}\}$. Let m index training episodes, distinct from the contact-point index t. Exploration is annealed linearly from $\epsilon_0 = 0.40$ to $\epsilon_{\min} = 0.10$ over the first 110 training episodes:

$$\epsilon(m) = \max\Big(0.10,\ 0.40 - 0.30\frac{m}{110}\Big). \tag{3}$$

The exploration rate remains at 0.10 thereafter. This schedule is identical across benchmarks and seeds and was verified against the stored checkpoints of all nine training runs.

Training budgets. We train three independent controller seeds per benchmark. Table 6 summarizes the training budgets and shared hyperparameters. Development tasks are revisited to accumulate controller experience, corresponding to approximately three, six, and eight passes on SWE-bench, Terminal-Bench, and τ^2-bench, respectively. Training and held-out evaluation tasks are disjoint; Appendix B.1 specifies the partitions.

Table 6: Controller-training budgets and shared hyperparameters. Evaluation uses a frozen greedy policy without exploration.

	SWE-bench	**Terminal-Bench**	**τ^2-bench**
Controller seeds	3	3	3
Training episodes	∼150	162	160
Training tasks	∼50	27	20
Approx. passes	3	6	8
Discount γ	1	1	1
MC update	$\alpha = 1/N(s,a)$		
ϵ_0	0.40		
$\epsilon_{\min}$	0.10		
Annealing	linear, first 110 episodes		
Evaluation ϵ	0 (greedy)		

Offline consultation-price re-estimation. The SWE-bench consultation-price ablation reuses the training trajectories collected under the $\lambda = 0.10$ behavior policy. For $\lambda \in \{0.35, 0.50\}$, we recompute the visit-specific returns with the corresponding consultation penalty, rebuild the sample-average value table, and evaluate the resulting greedy policy on fresh trajectories. This changes the consultation price without modifying either language model or collecting additional training episodes. The resulting tables are offline re-estimates under the original behavior-policy visitation distribution, not independently trained policies at each price.

Frozen deployment policy. After training or offline re-estimation, the value table is frozen. Evaluation selects the greedy action in each state with $\epsilon = 0$; ties, including entirely unvisited rows, resolve to SILENT. The episode-level consultation budget is enforced separately: once it is exhausted, subsequent CONSULT choices are forced to SILENT, as described in Appendix A.2. Deployment therefore requires only deterministic state extraction, table lookup, and budget enforcement, with no exploration or controller updates on held-out tasks.

B EXPERIMENTAL DETAILS

B.1 SPLIT CONSTRUCTION AND TASK SEPARATION

For each benchmark, controller development and final evaluation use disjoint task instances from the same benchmark family. The executor and advisor remain fixed throughout; only ARIA's consultation policy is learned on the development tasks. Final evaluation uses the same held-out task set across all compared methods.

Table 7 summarizes the resulting development and evaluation partitions.

Table 7: Development and evaluation splits used for policy learning and final evaluation. Train–test overlap was verified from task identifiers rather than assumed from split names.

Benchmark	**Development source**	**Train tasks**	**Train episodes**	**Held-out test**
SWE-bench	screened benchmark training pool	50-52	150-156	50 (Verified-Mini)
Terminal-Bench	39-task development partition	27	162	50 (test)
τ^2-bench	official telecom train split	20	160	40 (telecom test)

SWE-bench Verified. The development pool was constructed before controller training from a screening of 100 candidate tasks drawn from the SWE-bench Verified benchmark outside the Mini evaluation subset. Both underlying models were first run autonomously on these candidates. We retained two screening strata: (A) tasks solved by the *Executor-only* and (B) tasks solved by the *Advisor-only*. This initially produced 52 development tasks, comprising 21 tasks from stratum A and 31 from stratum B. The first controller seed used all 52 tasks; following unrecoverable rerun crashes, subsequent seeds were standardized to a 50-task development set. Tasks solved by neither model were excluded from controller development. The resulting development set is therefore not outcome-blind: its construction intentionally retains tasks on which autonomous execution is feasible or stronger-model capability provides a demonstrated advantage. The held-out 50-task Verified-Mini evaluation set was not used in this screening procedure, and its intersection with the 52-task development pool is empty.

Terminal-Bench 2.1. We first partitioned all 89 tasks into a 39-task development partition and a 50-task held-out test partition. The development partition was then filtered using six completed autonomous benchmark runs, comprising three runs with the executor model and three with the advisor model. Twelve development tasks that were not solved in any of these six runs were removed, leaving 27 controller development tasks. The held-out 50-task test partition was left unchanged. Each of the 27 development tasks is revisited for six passes, yielding 162 training episodes per controller seed.

This filtering is also not outcome-blind. In particular, it removes tasks that were never solved by either model from the development set while retaining such tasks in the held-out evaluation set. Consequently, the development set contains no never-solved tasks, whereas 15 of the 50 held-out tasks (30%) fall into this category. This makes the development distribution easier than the evaluation distribution and can increase consultation pressure when the frozen policy is deployed on the harder held-out tasks. The resulting asymmetry therefore does not favor lower evaluation cost.

τ^2-bench Telecom. We use 20 tasks from the official Telecom training split for controller training and the official 40-task Telecom test split verbatim for final evaluation. No evaluation task is used during controller learning.

Task separation and repeated development episodes. The development tasks are revisited to accumulate controller experience: approximately three passes on SWE-bench, six passes on Terminal-Bench, and eight passes on τ^2-bench. These repetitions provide additional state-action observations but do not introduce held-out test instances into policy learning.

We explicitly verified task identities after data preparation. The measured training-test overlap is zero for all three benchmarks: 0/50 held-out tasks for SWE-bench, 0/50 for Terminal-Bench, and

0/40 for τ^2-bench. Thus, all reported final-evaluation results are obtained on task instances that were not used to derive the frozen consultation tables.

Training episodes that terminate before reaching a terminal task outcome do not update the controller, consistent with Appendix A.3. During final evaluation, failed or crashed episodes remain part of the benchmark denominator even when no usable trajectory is available for subsequent diagnostic analyses.

B.2 Baseline Contracts

All evaluation conditions use the same held-out task instances, task order, executor–advisor model pair, workflow implementation, and benchmark harness unless the allocation method itself requires different execution semantics. The lower-cost executor is `gemini-3-flash-preview` at low reasoning effort, and the stronger model is `gemini-3.1-pro-preview` at high reasoning effort. Within-trajectory consultation conditions use the common advisor interface defined in Appendix A.1 unless stated otherwise.

Headline consultation conditions use an episode-level budget of $B = 4$ advisor calls. *Plan* consults only at the first eligible opportunity and therefore has an effective budget of $B = 1$. Capability anchors and task-level allocation baselines do not use this within-trajectory consultation budget. Table 8 summarizes the resulting contracts.

Table 8: Baseline contracts. B denotes the maximum number of within-trajectory advisor consultations per episode and therefore applies only to consultation-based conditions.

Method	**Allocation rule**	**Budget**
Executor-only	no advisor	–
Advisor-only	advisor executes	–
FrugalGPT	score-based cascade	–
RouteLLM	pre-task routing	–
AutoMix	verify then escalate	–
Plan	first contact	$B = 1$
Heuristic	beginning/end triggers	$B = 4$
Always	at every contact point	$B = 4$
LLM-Router	orchestrator agent selects advisor	$B = 4$
Random	rate-matched advice	$B = 4$
ARIA	state-conditioned policy	$B = 4$

Capability anchors. *Executor-only* and *Advisor-only* define the two autonomous reference points used throughout the evaluation. Under *Executor-only*, the lower-cost executor completes the workflow without access to advisor guidance. Under *Advisor-only*, the stronger model replaces the executor and completes the full workflow itself. These conditions therefore vary model capability rather than consultation placement and define the success and cost anchors used for PGR, CostShare, and ER.

Task-level allocation baselines. *FrugalGPT*, *RouteLLM*, and *AutoMix* allocate model capability at task level rather than at recurring workflow contact points. *FrugalGPT* implements a score-based cascade: every task first incurs lower-cost executor execution, and tasks selected for escalation additionally incur stronger-model execution. *RouteLLM* performs routing before agentic execution and selects exactly one of the two models to execute the task. *AutoMix* first executes the task with the lower-cost model, applies its self-verification step, and invokes stronger-model execution for tasks that trigger escalation. The self-verification pass is metered separately and included in the reported *AutoMix* cost. These conditions test whether task-level routing or cascading is sufficient without conditioning allocation on evidence revealed later in the trajectory.

Fixed consultation schedules. *Plan*, *Heuristic*, and *Always* use the same executor–advisor intervention contract as ARIA but replace its learned allocation rule with fixed schedules. *Plan* consults only at the first eligible contact point and therefore tests front-loaded advisor access. *Heuristic*

uses fixed beginning/end and stall-related triggers. *Always* requests advice at every eligible contact point until the episode-level budget is exhausted, providing the high-consultation reference under the shared $B = 4$ cap.

Native orchestrator routing. *LLM-Router* removes ARIA's learned consultation policy and lets the native LLM orchestrator decide when to involve the advisor. In the underlying workflow, the orchestrator already selects which available agent acts next; for this condition, the advisor is included among those selectable agents. The orchestrator therefore receives repeated opportunities to involve the advisor during execution, with each decision informed by the full task ledger available at that point. It does not use ARIA's compact state abstraction or learned value table. Advisor selections count against the same episode-level budget of $B = 4$ used by the other headline consultation conditions. *LLM-Router* thus tests whether the workflow's native LLM orchestrator can determine when to involve the advisor without a separately learned consultation controller.

Placement control. *Random* replaces ARIA's state-conditioned action with an independent Bernoulli decision at every eligible contact point while retaining the same advisor intervention semantics and episode-level budget. Its probability is calibrated before running the control from the corresponding fixed ARIA reference arm to approximate its consultation volume. The complete calibration procedure and realized matching rates are reported in Appendix B.3.

Learned consultation. ARIA uses the frozen greedy consultation policy learned on the development tasks. At each eligible contact point, it maps the observable workflow state to SILENT or CONSULT; ties, including unvisited states, resolve to SILENT. The advisor interface itself is identical to that used by *Plan*, *Heuristic*, *Always*, and *Random*. These conditions therefore differ in the rule that determines where consultation is allocated rather than in the form of advice delivered.

Budget sensitivity. For the separate budget-sensitivity analysis in Section 5.1, we additionally evaluate *Always* with $B = 10$ on all three benchmarks. These variants are reported separately and are not part of the common-budget headline comparison.

B.3 RANDOM CALIBRATION

Random serves as a placement control for ARIA. At each eligible contact point, it consults the advisor independently with a fixed probability p, subject to the same episode-level budget of $B = 4$. Unlike ARIA, the decision does not depend on the current state or trajectory. The probability p is fixed before evaluating *Random* and is chosen to approximately match the consultation volume of the corresponding headline ARIA policy ($\lambda = 0.10$).

Let K_i denote the number of consultations made by the ARIA reference policy in episode i, and let C_i denote the number of eligible contact points in that episode. The nominal consultation probability is

$$p = \frac{\sum_i K_i}{\sum_i C_i}. \tag{4}$$

The resulting value is rounded to two decimal places, or three when $p < 0.25$, and is then held fixed throughout evaluation.

For Terminal-Bench, the $B = 4$ consultation cap can noticeably reduce the number of consultations realized from independent draws. In addition, complete contact-point logs are unavailable for 25 of the 150 crash-path episodes. We therefore use reconstructed contact-point counts and choose p so that the expected capped consultation volume matches the ARIA reference:

$$\sum_i \mathbb{E}[\min(\mathrm{Bin}(C_i, p), B)] = \sum_i K_i. \tag{5}$$

Table 9 reports the resulting probabilities and realized consultation volumes.

Table 9: Calibration and realized consultation volume of the Random placement control relative to the headline ARIA policy ($\lambda = 0.10$).

	SWE-bench	**Terminal-Bench**	**τ^2-bench**
ARIA consults/episode	2.780	2.460	3.042
Consultation probability p	0.49	0.62	0.172
Random consults/episode	2.527	2.417	2.670
Random / ARIA volume	90.9%	98.2%	87.8%

We consider consultation volume matched when the realized number of consultations per episode is within $\pm 10\%$ of the ARIA reference. SWE-bench and Terminal-Bench satisfy this criterion. On τ^2-bench, *Random* reaches 87.8% of the ARIA consultation volume and is therefore treated as approximately rate-matched rather than exactly matched.

We match consultation volume rather than dollar cost because consultation timing can change the subsequent trajectory and therefore the amount of context processed by later model calls. Realized model cost is reported separately. The probability p is not adjusted after observing the *Random* evaluation trajectories.

B.4 Token Usage and Billing

To make the reported cost comparisons independently auditable, we reconstruct the headline evaluation costs from the raw billing artifacts and reconcile them against both the episode ledgers and the values reported in Table 2. The scope is restricted to the headline configuration: three evaluation seeds for each of the 33 benchmark-condition cells, with the common $B = 4$ consultation budget where applicable. Training runs, pilot experiments, the $\lambda \in \{0.35, 0.50\}$ ablations, and the $B = 10$ budget-sensitivity arms are excluded. All 33 headline cells are recovered, with reconstructed mean costs differing from Table 2 by at most \$0.0049.

Pricing and token accounting. Each model call records total prompt tokens, cached prompt tokens, completion tokens, reasoning tokens, and billed cost. Because the prompt count includes the cached portion, fresh input is

$$f = n_{\text{prompt}} - n_{\text{cached}}.$$

For a call with fresh input f, cached input k, completion tokens o, and reasoning tokens r, the base cost is

$$c = \frac{f p_{\text{in}} + k p_{\text{cached}} + (o + r) p_{\text{out}}}{10^6}.$$

Reasoning tokens are therefore billed in addition to completion tokens at the output-token rate. This interpretation reproduces the logged provider cost for all 106,662 non-failed priced calls in the reconstruction; treating reasoning tokens as already contained in the completion count does not.

The pricing schedule is read from the frozen experiment configuration and mirrored into each run manifest. For `gemini-3.1-pro-preview`, calls whose prompt exceeds 200K tokens incur the long-context surcharge shown in Table 10. The surcharge is applied per call after the base logger calculation.

Model	Input	Cached input	Output	Reasoning	Tier threshold	Above threshold
`gemini-3-flash-preview`	0.50	0.05	3.00	3.00	none	none
`gemini-3.1-pro-preview`	2.00	0.20	12.00	12.00	200,000 prompt tokens	in 2.00 / cached 0.20 / out 6.00

Table 10: Pricing used for the reported evaluation costs, in USD per million tokens. The final column reports the incremental surcharge applied above the listed context threshold. Thus, for `gemini-3.1-pro-preview`, the effective rates above 200K prompt tokens are \$4.00 for fresh input, \$0.40 for cached input, and \$18.00 for output and reasoning tokens.

Episode and condition attribution. A re-run episode may leave multiple attempts in the billing log while the episode ledger retains only the attempt used for evaluation. We identify the attributed

attempt by matching the sum of its call costs to the corresponding ledger cost rather than by file order. Within the headline audit scope, 4,066 unique episode records contain at least one call record. Of these, 4,052 contain a single matching attempt and 14 contain multiple attempts with exactly one match; no case remains unresolved.

The accounting also follows the execution semantics of each condition. *FrugalGPT* and *AutoMix* are cascades and therefore incur the lower-cost executor run for every task plus stronger-model execution for the tasks they escalate. *RouteLLM* selects exactly one executor per task. The *AutoMix* self-verification pass is metered separately from the agent call log and is included in its reported cost. Repeated task runs are resolved according to the frozen campaign precedence before aggregation.

Benchmark-side model usage. The τ^2 user simulator is metered separately from the evaluated agent. It is part of the benchmark environment and is not included in the model costs reported in Table 2. Across the reconstructed τ^2 source runs, this simulator accounts for 24,826 calls, 110.6M recorded tokens, and $48.26. Its meter does not expose cached-input tokens, so cached simulator usage is reported as unavailable rather than inferred or treated as zero.

Token usage. Table 11 reports the complete token breakdown used for the headline conditions. Fresh input and cached input are shown separately, and reasoning tokens are reported separately from ordinary completion tokens. All token quantities are summed across the three evaluation seeds, whereas the final cost column reports the mean evaluation-run cost used in Table 2.

For cascade conditions, the component-episode count includes both the base execution and any escalated execution and should therefore not be interpreted as the number of unique benchmark tasks. Likewise, rows in Table 11 must not be summed to obtain a campaign total: some conditions deliberately reuse and re-charge the same lower-cost base runs.

Benchmark	Condition	Ep.	Calls	Input	Cached	Output	Reason.	Total	Cached %	Cost/run
SWE-bench Verified	Executor-only	150	2,440	36.825	12.397	4.251	0.880	54.354	25.2	11.48
SWE-bench Verified	Advisor-only	150	3,367	28.609	43.244	2.164	2.886	76.903	60.2	45.90
SWE-bench Verified	FrugalGPT	196	3,470	43.958	21.112	4.850	1.675	71.596	32.5	22.61
SWE-bench Verified	RouteLLM	150	2,734	29.913	21.392	3.524	1.924	56.753	41.7	25.03
SWE-bench Verified	AutoMix	257	4,945	66.469	66.846	5.954	3.133	144.438	50.1	59.68
SWE-bench Verified	Plan	150	2,736	31.402	9.571	3.782	0.435	45.190	23.4	10.90
SWE-bench Verified	Heuristic	150	3,453	47.571	43.610	4.446	1.655	97.281	47.8	21.50
SWE-bench Verified	Always	150	3,827	44.080	38.790	3.908	2.110	88.887	46.8	21.39
SWE-bench Verified	LLM-Router	150	2,870	46.561	37.974	4.950	1.145	90.630	44.9	15.77
SWE-bench Verified	Random	150	3,320	38.589	23.170	4.085	1.485	67.329	37.5	17.33
SWE-bench Verified	ARIA	150	3,488	46.150	39.643	4.729	1.585	92.107	46.2	22.09
Terminal-Bench 2.1	Executor-only	150	1,827	31.581	17.717	3.512	0.347	53.157	35.9	9.76
Terminal-Bench 2.1	Advisor-only	150	3,332	36.416	83.526	1.860	3.658	125.460	69.6	60.99
Terminal-Bench 2.1	FrugalGPT	154	1,919	32.161	18.780	3.586	0.493	55.020	36.9	11.10
Terminal-Bench 2.1	RouteLLM	150	2,330	36.852	29.530	3.272	1.230	70.883	44.5	21.66
Terminal-Bench 2.1	AutoMix	159	2,178	35.608	24.130	3.821	0.822	66.480	40.4	16.51
Terminal-Bench 2.1	Plan	150	2,450	30.332	25.749	3.932	0.907	60.920	45.9	11.34
Terminal-Bench 2.1	Heuristic	150	3,106	55.478	55.717	3.928	2.356	117.479	50.1	27.47
Terminal-Bench 2.1	Always	150	3,373	59.327	50.774	4.183	1.979	116.263	46.1	27.94
Terminal-Bench 2.1	LLM-Router	150	2,809	49.815	43.169	4.947	1.124	99.055	46.4	18.47
Terminal-Bench 2.1	Random	150	2,870	48.688	42.470	3.571	1.490	96.219	46.6	22.25
Terminal-Bench 2.1	ARIA	150	3,356	51.080	57.531	3.811	1.993	114.415	53.0	24.93
τ^2-bench Telecom	Executor-only	120	4,852	21.605	25.813	0.978	0.058	48.455	54.4	5.07
τ^2-bench Telecom	Advisor-only	120	4,288	18.024	22.365	0.785	1.462	42.635	55.4	22.50
τ^2-bench Telecom	FrugalGPT	127	5,232	23.148	28.187	1.048	0.179	52.563	54.9	7.02
τ^2-bench Telecom	RouteLLM	120	4,780	20.128	27.370	0.953	0.444	48.895	57.6	9.61
τ^2-bench Telecom	AutoMix	218	8,665	37.079	47.209	1.688	1.312	88.611	56.0	24.98
τ^2-bench Telecom	Plan	120	4,771	21.710	25.462	0.952	0.122	48.245	54.0	5.69
τ^2-bench Telecom	Heuristic	120	4,764	21.926	25.933	0.933	0.268	49.059	54.2	7.13
τ^2-bench Telecom	Always	120	5,121	23.072	29.640	0.991	0.458	54.162	56.2	9.01
τ^2-bench Telecom	LLM-Router	120	5,106	23.877	28.333	1.037	0.101	53.348	54.3	6.16
τ^2-bench Telecom	Random	120	4,905	21.537	28.631	0.958	0.478	51.604	57.1	8.24
τ^2-bench Telecom	ARIA	120	4,897	21.359	28.469	0.955	0.434	51.217	57.1	7.77

Table 11: Token usage and realized model cost for all headline evaluation conditions. Token columns are totals across three evaluation seeds in millions of tokens. "Input" denotes fresh input excluding cached prompt tokens; "Cached %" is the cached share of total prompt tokens. "Cost/run" is the mean realized cost across the three evaluation seeds, corresponding to Table 2. For cascades, the episode column counts billed component executions rather than unique benchmark tasks.

Cost reconciliation. The reconstructed accounting reproduces all 33 headline cost cells. The maximum absolute difference between the reconstructed mean and the value printed in Table 2 is \$0.0049, and the maximum deviation from the signed-off per-seed costs is \$0.0006. The per-call pricing equation itself reproduces 100% of the 106,662 non-failed priced calls for which complete token records are available.

Accounting completeness. A small number of episodes lack complete per-call token records. In these cases, reported costs are retained from the episode-level ledgers rather than inferred from token counts. Consequently, condition-level costs remain unchanged, while token aggregates include only recorded per-call usage.

C Additional Results and Analyses

C.1 Seed-Level Dispersion

To quantify run-to-run variation, Table 12 reports sample standard deviations across the three evaluation seeds corresponding to the means in Table 2. These values describe dispersion across repeated runs rather than confidence intervals over the underlying task distribution.

Variation is not specific to ARIA. Its success standard deviation is 3.5, 5.0, and 2.9 percentage points on SWE-bench, Terminal-Bench, and τ^2-bench, respectively, while several comparison conditions exhibit equal or larger variation. We therefore interpret small differences between nearby operating points descriptively rather than as evidence of statistical separation.

Table 12: Task success and realized model cost across three evaluation seeds. Entries report mean $\pm$ sample standard deviation ($n = 3$).

	SWE-bench Verified		**Terminal-Bench 2.1**		**τ^2-bench Telecom**	
Condition	Succ. (%)	Cost ($)	Succ. (%)	Cost ($)	Succ. (%)	Cost ($)
Executor-only	18.0 ± 4.0	11.48 ± 2.30	21.3 ± 3.1	9.76 ± 1.90	72.5 ± 4.3	5.07 ± 0.04
Advisor-only	62.0 ± 7.2	45.90 ± 5.30	51.3 ± 1.2	60.99 ± 5.05	98.3 ± 2.9	22.50 ± 1.22
FrugalGPT	31.3 ± 2.3	22.61 ± 1.44	22.0 ± 4.0	11.10 ± 2.89	78.3 ± 2.9	7.02 ± 0.40
RouteLLM	30.0 ± 5.3	25.03 ± 2.35	28.0 ± 3.5	21.66 ± 0.85	88.3 ± 3.8	9.61 ± 0.12
AutoMix	52.0 ± 3.5	59.68 ± 5.75	22.0 ± 2.0	16.51 ± 4.21	97.5 ± 2.5	24.98 ± 0.64
Plan	24.0 ± 7.2	10.90 ± 0.42	24.0 ± 5.3	11.34 ± 0.67	70.0 ± 4.3	5.69 ± 0.27
Heuristic	41.3 ± 3.1	21.50 ± 1.35	34.7 ± 3.1	27.47 ± 2.76	83.3 ± 5.2	7.13 ± 0.10
Always	34.0 ± 2.0	21.39 ± 0.88	35.3 ± 4.2	27.94 ± 2.68	87.5 ± 2.5	9.01 ± 0.12
LLM-Router	29.3 ± 4.2	15.77 ± 1.76	30.0 ± 2.0	18.47 ± 0.96	87.5 ± 4.3	6.16 ± 0.34
Random	28.0 ± 2.0	17.33 ± 2.11	30.7 ± 5.0	22.25 ± 3.07	85.8 ± 5.2	8.24 ± 0.51
ARIA	48.0 ± 3.5	22.09 ± 2.65	34.7 ± 5.0	24.93 ± 2.73	93.3 ± 2.9	7.77 ± 1.47

C.2 Placement Analysis

Section 5.1 compares ARIA with *Heuristic* and *Random* to separate consultation placement from advisor access and consultation volume. Figure 4 visualizes this comparison by plotting task success against the realized number of consultations per episode. On SWE-bench, the three conditions operate at similar consultation volumes but occupy clearly different success levels, with ARIA above both fixed and uninformed placement. The separation is smaller on Terminal-Bench, where ARIA and *Heuristic* reach the same mean success despite allocating advice differently. On τ^2-bench, ARIA again attains higher success than both comparison conditions, although *Random* realizes somewhat less advice than the target volume. The figure therefore complements the aggregate success–cost results by showing how consultation outcomes vary with where, rather than only how often, advisor access is allocated.

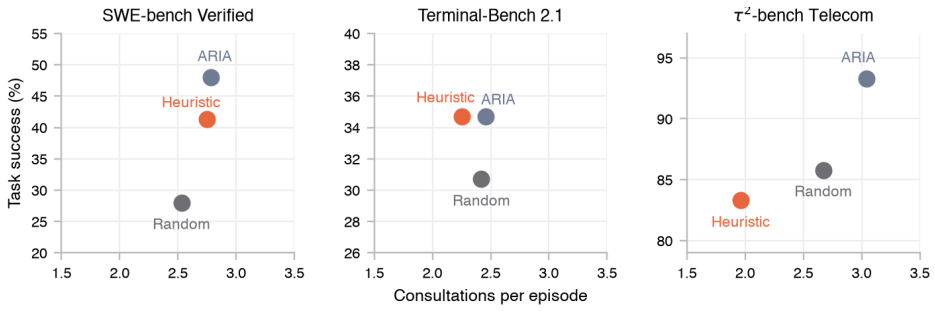

Figure 4: Task success and consultation frequency for ARIA, *Heuristic*, and *Random* across all three benchmarks.

C.3 Training Dynamics and Policy Inspection

We examine the controller through two complementary views: its behavior during training and the action values underlying its frozen decisions.

Training dynamics. Figure 5 compares consultation frequency with changes in the greedy policy across the three training seeds of each benchmark. On τ^2-bench, consultation rates remain comparatively similar across seeds even while individual greedy decisions continue to change. SWE-bench and Terminal-Bench show greater variation in consultation frequency. The two views therefore capture different properties: a stable aggregate consultation rate does not imply an unchanged state-conditioned policy. These plots describe observed training behavior rather than establish policy convergence.

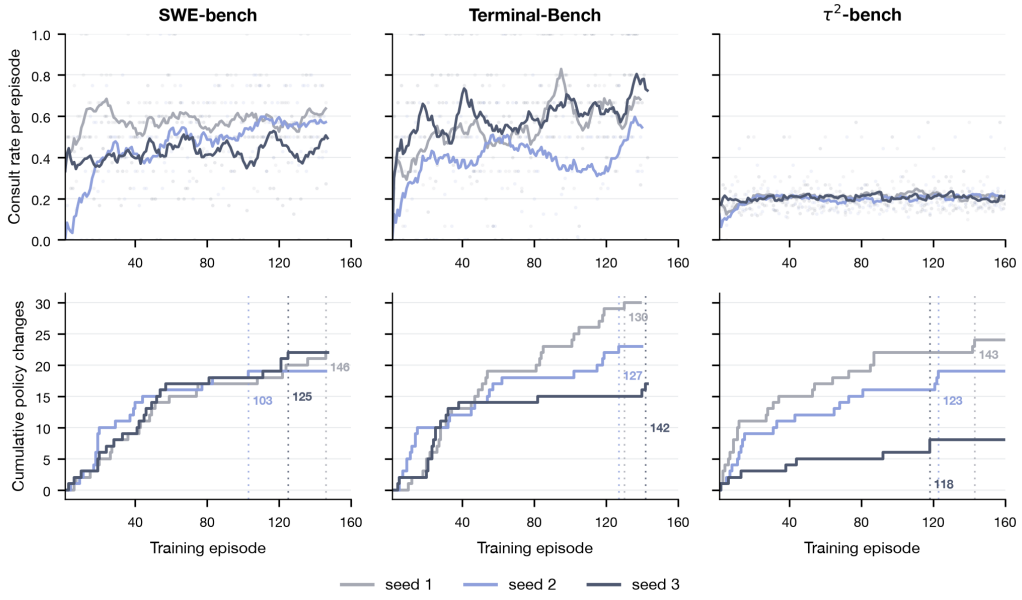

Figure 5: Training dynamics across three benchmarks and three controller seeds. **Top:** per-episode consultation rates (faint points) and smoothed trajectories (solid lines). **Bottom:** cumulative changes in the greedy tabular policy. Dotted vertical lines mark the last observed greedy-policy change for each seed, not a certified convergence point.

Inspecting the frozen policies. Figures 6–8 report the learned action values and state visit counts for each benchmark and controller seed, together with corresponding heatmap visualizations. The Q-values are empirical estimates of cost-aware return rather than calibrated success probabilities. For each state, the greedy policy compares $Q(s, \textsc{Consult})$ with $Q(s, \textsc{Silent})$; the larger value determines the preferred action, subject to the tie-breaking rule and budget constraint in Appendix A.3. The relevant comparison is therefore between actions within the same state rather than between absolute values across benchmarks or seeds.

The compact tabular representation makes each consultation decision directly inspectable, while the heatmaps expose broader allocation patterns. Table 13 complements these per-seed policies by summarizing agreement in their resulting greedy actions across seeds.

Seed 1

State	N	Q_S/Q_C
F/N/0	149	0.033/-0.007
E/N/0	38	0.095/0.017
E/N/1	171	0.041/0.011
E/C/0	1	0.029/0.000
E/C/1	22	-0.075/0.109
E/X/0	1	0.029/0.000
E/X/1	22	-0.116/0.269
R/N/0	16	0.258/-0.228
R/N/1	43	-0.077/-0.112
R/C/0	6	-0.142/0.000
R/C/1	86	0.152/0.010
R/X/0	0	0.000/0.000
R/X/1	47	-0.266/-0.031
D/C/0	9	-0.912/0.311
D/C/1	129	-0.100/0.095
D/X/0	8	0.000/-0.044
D/X/1	81	-0.028/0.159

Seed 2

State	N	Q_S/Q_C
F/N/0	149	0.007/-0.055
E/N/0	113	-0.065/0.029
E/N/1	97	0.028/0.009
E/C/0	8	-0.426/0.091
E/C/1	12	-0.279/0.168
E/X/0	5	0.418/0.000
E/X/1	15	0.169/-0.045
R/N/0	1	0.088/0.000
R/N/1	44	0.064/-0.096
R/C/0	1	-0.882/0.000
R/C/1	88	0.049/-0.108
R/X/0	0	0.000/0.000
R/X/1	68	-0.277/0.148
D/C/0	16	-0.338/0.224
D/C/1	115	0.088/0.129
D/X/0	21	-0.533/-0.017
D/X/1	88	-0.079/0.011

Seed 3

State	N	Q_S/Q_C
F/N/0	149	0.014/-0.147
E/N/0	173	0.047/-0.135
E/N/1	35	-0.093/-0.170
E/C/0	14	-0.243/-0.178
E/C/1	12	0.055/-0.311
E/X/0	5	-0.515/0.375
E/X/1	14	0.467/0.153
R/N/0	35	0.078/-0.431
R/N/1	16	-0.213/-0.412
R/C/0	37	0.125/0.157
R/C/1	86	-0.101/0.042
R/X/0	5	-0.022/-1.000
R/X/1	39	-0.103/0.127
D/C/0	41	-0.279/0.110
D/C/1	81	-0.118/0.036
D/X/0	48	-0.434/0.192
D/X/1	57	-0.179/0.037

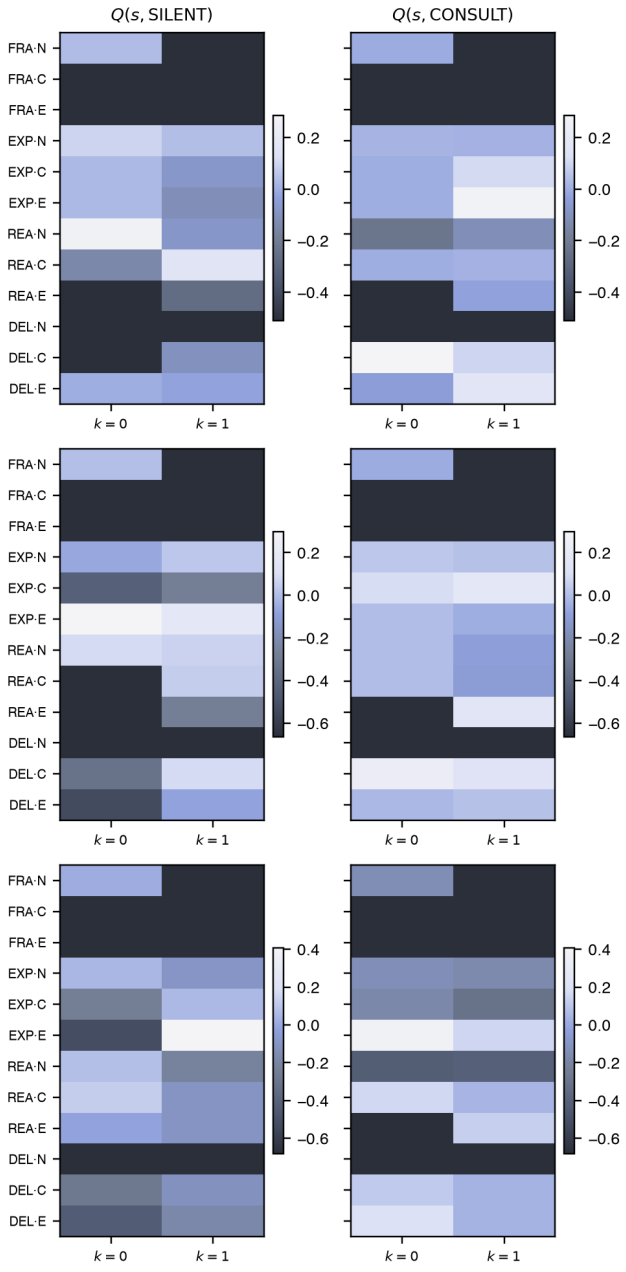

Figure 6: Learned action values for SWE-bench Verified across three independently trained controller seeds. **Left:** compact numerical tables reporting state visit counts and action values, where $Q_S = Q(s, \text{SILENT})$ and $Q_C = Q(s, \text{CONSULT})$. State notation is *phase/evidence/prior consultation*: F/E/R/D denote FRAME, EXPLORE, REASON, and DELIVER; N/C/X denote NONE, CLEAN, and ERROR. Structurally unreachable states are omitted, while reachable states with $N = 0$ are retained. **Right:** the corresponding learned value tables for the three seeds visualized as heatmaps. Deployment selects the action with the larger value within each state; ties resolve to SILENT.

Seed 1

State	N	Q_S/Q_C
F/N/0	139	0.246/0.434
E/N/0	2	1.000/0.000
E/N/1	2	-0.073/0.000
E/C/0	6	0.476/-0.049
E/C/1	90	0.342/0.107
E/X/0	2	-0.049/-0.049
E/X/1	45	0.331/0.341
R/N/0	1	1.000/0.000
R/N/1	57	-0.013/-0.049
R/C/0	1	1.000/0.000
R/C/1	188	0.028/0.242
R/X/0	0	0.000/0.000
R/X/1	105	0.002/0.224
D/C/0	5	0.750/-0.049
D/C/1	137	0.484/0.512
D/X/0	1	0.000/0.000
D/X/1	34	0.000/0.236

Seed 2

State	N	Q_S/Q_C
F/N/0	140	0.483/0.257
E/N/0	0	0.000/0.000
E/N/1	0	0.000/0.000
E/C/0	68	0.309/0.417
E/C/1	43	0.330/0.151
E/X/0	23	0.368/0.186
E/X/1	13	0.142/0.237
R/N/0	0	0.000/0.000
R/N/1	0	0.000/0.000
R/C/0	1	0.000/0.000
R/C/1	413	0.043/0.007
R/X/0	1	-0.024/0.000
R/X/1	28	-0.022/-0.058
D/C/0	45	0.625/0.722
D/C/1	109	0.286/0.467
D/X/0	18	0.167/0.460
D/X/1	20	0.143/-0.032

Seed 3

State	N	Q_S/Q_C
F/N/0	143	0.537/0.540
E/N/0	0	0.000/0.000
E/N/1	5	-0.073/-0.073
E/C/0	11	0.339/0.000
E/C/1	89	0.495/0.393
E/X/0	9	0.000/0.641
E/X/1	40	0.619/0.512
R/N/0	0	0.000/0.000
R/N/1	57	-0.026/-0.049
R/C/0	0	0.000/0.000
R/C/1	177	0.043/0.253
R/X/0	0	0.000/0.000
R/X/1	49	-0.007/0.191
D/C/0	9	0.667/0.594
D/C/1	152	0.290/0.700
D/X/0	5	0.200/0.000
D/X/1	27	0.071/0.409

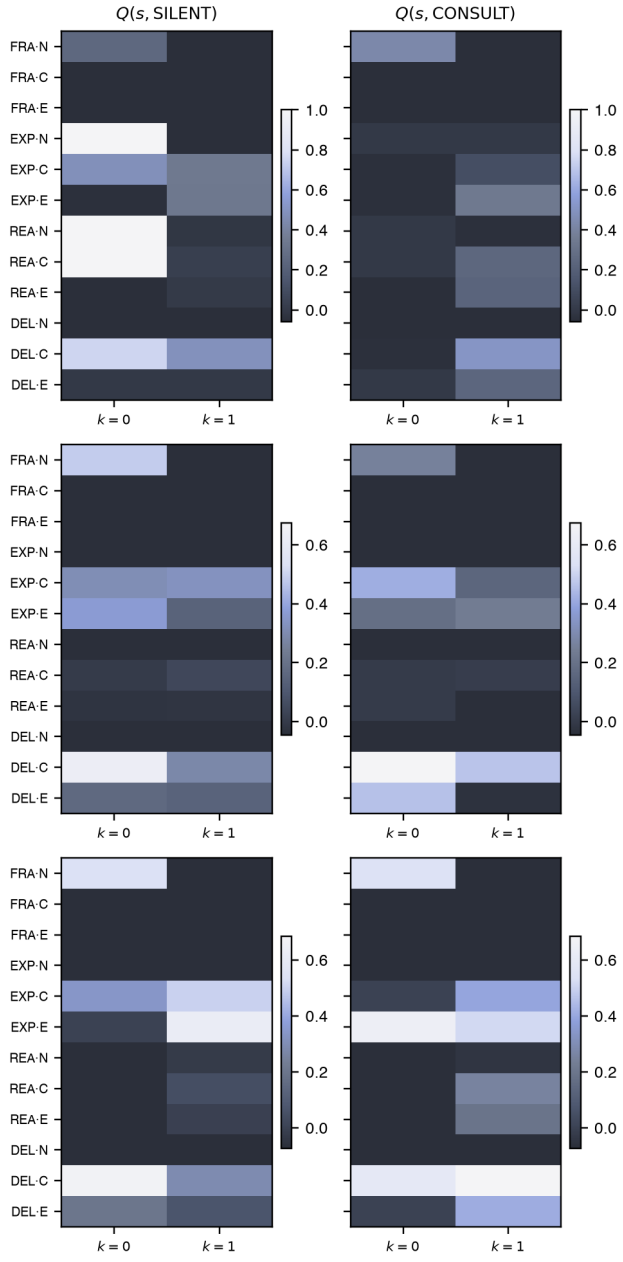

Figure 7: Learned action values for Terminal-Bench 2.1 across three independently trained controller seeds. **Left:** compact numerical tables reporting state visit counts and action values, where $Q_S = Q(s, \text{SILENT})$ and $Q_C = Q(s, \text{CONSULT})$. State notation is *phase/evidence/prior consultation*: F/E/R/D denote FRAME, EXPLORE, REASON, and DELIVER; N/C/X denote NONE, CLEAN, and ERROR. Structurally unreachable states are omitted, while reachable states with $N = 0$ are retained. **Right:** the corresponding learned value tables for the three seeds visualized as heatmaps. Deployment selects the action with the larger value within each state; ties resolve to SILENT.

Seed 1

State	N	Q_S/Q_C
F/N/0	160	0.733/0.555
E/N/0	167	0.673/0.757
E/N/1	74	0.647/0.543
E/O/0	27	0.674/0.750
E/O/1	52	0.745/0.848
E/F/0	0	0.000/0.000
E/F/1	0	0.000/0.000
R/N/0	1	-0.076/0.000
R/N/1	4	0.943/0.000
R/O/0	51	0.393/0.638
R/O/1	2641	0.745/0.749
R/F/0	0	0.000/0.000
R/F/1	0	0.000/0.000
D/O/0	1	0.000/0.000
D/O/1	159	0.792/0.000
D/F/0	0	0.000/0.000
D/F/1	0	0.000/0.000

Seed 2

State	N	Q_S/Q_C
F/N/0	160	0.512/0.690
E/N/0	30	0.465/0.433
E/N/1	223	0.701/0.678
E/O/0	7	0.803/0.000
E/O/1	60	0.779/0.722
E/F/0	0	0.000/0.000
E/F/1	0	0.000/0.000
R/N/0	1	0.924/0.000
R/N/1	5	0.943/0.000
R/O/0	30	0.137/0.554
R/O/1	2711	0.694/0.708
R/F/0	0	0.000/0.000
R/F/1	4	1.000/0.000
D/O/0	1	0.000/0.000
D/O/1	158	0.740/0.856
D/F/0	0	0.000/0.000
D/F/1	1	1.000/0.000

Seed 3

State	N	Q_S/Q_C
F/N/0	160	0.599/0.745
E/N/0	21	0.568/0.424
E/N/1	223	0.769/0.415
E/O/0	7	0.760/0.924
E/O/1	69	0.839/0.610
E/F/0	0	0.000/0.000
E/F/1	0	0.000/0.000
R/N/0	2	0.943/0.000
R/N/1	11	-0.038/0.843
R/O/0	91	0.662/0.656
R/O/1	2624	0.747/0.784
R/F/0	0	0.000/0.000
R/F/1	0	0.000/0.000
D/O/0	1	0.000/0.000
D/O/1	159	0.809/0.981
D/F/0	0	0.000/0.000
D/F/1	0	0.000/0.000

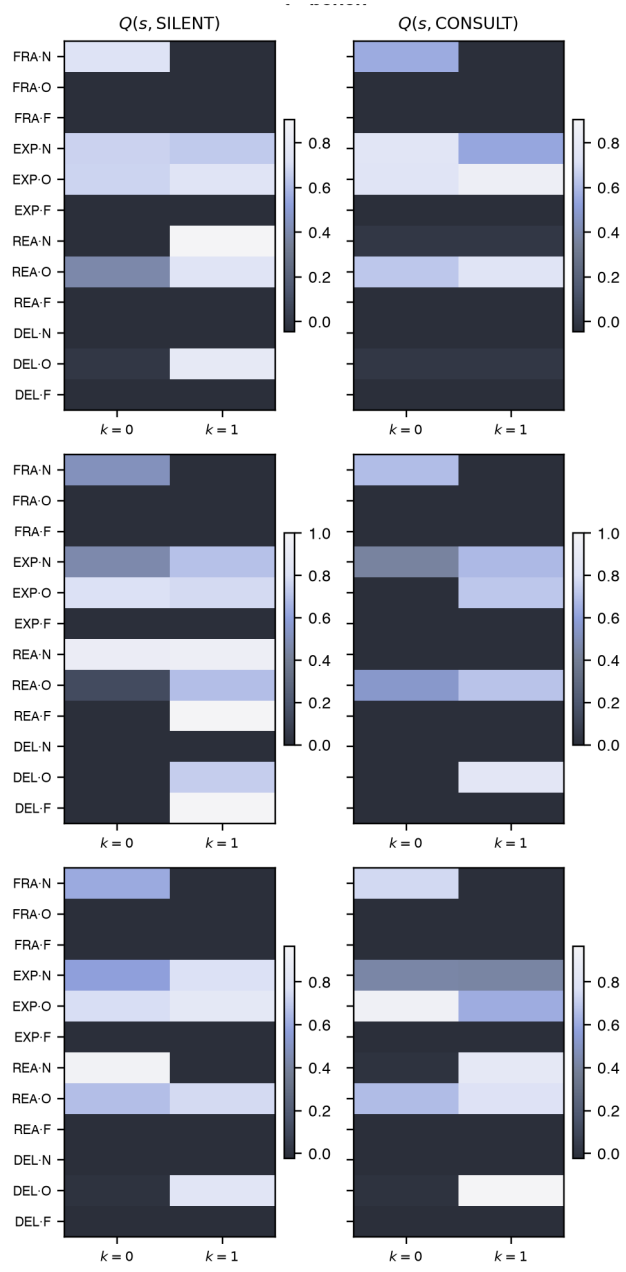

Figure 8: Learned action values for τ^2-bench Telecom across three independently trained controller seeds. **Left:** compact numerical tables reporting state visit counts and action values, where $Q_S = Q(s, \text{SILENT})$ and $Q_C = Q(s, \text{CONSULT})$. State notation is *phase/evidence/prior consultation*: F/E/R/D denote FRAME, EXPLORE, REASON, and DELIVER; N/O/F denote NONE, OK, and FAILED. Structurally unreachable states are omitted, while reachable states with $N = 0$ are retained. **Right:** the corresponding learned value tables for the three seeds visualized as heatmaps. Deployment selects the action with the larger value within each state; ties resolve to SILENT.

C.4 POLICY CONSENSUS

Table 13 reports the greedy consultation decisions learned independently across the three seeds of each benchmark. The All column aggregates all nine frozen policies for direct cross-benchmark comparison; unvisited states contribute the deployment default SILENT. Agreement is limited overall, reflecting benchmark-specific allocation, but one state recurs across nearly all policies: at DELIVER with clean/OK evidence after prior consultation, eight of nine frozen tables select CONSULT.

Table 13: Cross-seed agreement of the learned greedy consultation policies. Entries report the number of seeds whose frozen table selects CONSULT in a given state. A dash indicates that the state was never visited during training for that benchmark. For τ^2, OK and FAILED correspond to the CLEAN and ERROR evidence classes used on SWE-bench and Terminal-Bench.

Position	Evidence	Prior consult	SWE	TB	τ^2	All
Frame	None	No	0/3	2/3	2/3	4/9
Explore	None	No	1/3	0/3	1/3	2/9
	None	Yes	0/3	1/3	0/3	1/9
	Clean/OK	No	2/3	1/3	2/3	5/9
	Clean/OK	Yes	2/3	0/3	1/3	3/9
	Error/Fail	No	1/3	1/3	–	2/9
	Error/Fail	Yes	1/3	2/3	–	3/9
Reason	None	No	0/3	0/3	1/3	1/9
	None	Yes	0/3	0/3	1/3	1/9
	Clean/OK	No	3/3	0/3	2/3	5/9
	Clean/OK	Yes	1/3	2/3	3/3	6/9
	Error/Fail	No	0/3	1/3	–	1/9
	Error/Fail	Yes	3/3	2/3	0/3	5/9
Deliver	Clean/OK	No	3/3	1/3	0/3	4/9
	Clean/OK	**Yes**	**3/3**	**3/3**	**2/3**	**8/9**
	Error/Fail	No	2/3	1/3	–	3/9
	Error/Fail	Yes	3/3	2/3	0/3	5/9

C.5 WITHIN-STATE CONSULTATION EFFECTS

The learned value table estimates consultation value from the remaining episodic return. As a complementary analysis, we use the randomized exploration during controller training to estimate how consultation changes outcomes within individual state cells. This analysis is distinct from the consultation-price sweep in Section 5.2: the price sweep re-estimates a policy from the same experience under a different reward penalty, whereas the analysis below compares CONSULT and SILENT decisions taken within the same abstract state.

Estimator. During training, the ϵ-greedy behavior policy logs the probability of consultation for every decision. Because ϵ is bounded below by 0.10, the minority action has probability at least 0.05, bounding the largest inverse-propensity weight at 20. Decisions forced by initialization or by an exhausted consultation budget contain no randomized action choice and are excluded.

For each state cell s, we estimate the self-normalized Hájek contrast

$$\hat{\tau}_s = \frac{\sum_{i:a_i=1} w_i Y_i}{\sum_{i:a_i=1} w_i} - \frac{\sum_{i:a_i=0} w_i Y_i}{\sum_{i:a_i=0} w_i}, \qquad w_i = \begin{cases} 1/p_i, & a_i = \text{CONSULT}, \\ 1/(1-p_i), & a_i = \text{SILENT}, \end{cases}$$

where p_i is the logged consultation propensity and Y_i the corresponding outcome. No weight trimming is applied. Uncertainty is estimated with 10,000 bootstrap replicates resampling complete episodes rather than individual decisions, preserving dependence among decisions from the same trajectory. Cells with fewer than five observations for either action are not assigned an effect estimate.

After these exclusions, the analysis contains 2,373 decisions from 448 SWE-bench episodes, 2,115 decisions from 429 Terminal-Bench episodes, and 4,541 decisions from 480 τ^2-telecom episodes.

In Figure 9, $k = 0$ denotes no previous consultation and $k \geq 1$ denotes at least one previous consultation, corresponding to $h_t = 0$ and $h_t = 1$ in the state representation.

Outcome definition. The primary outcome is whether the workflow step gated by the decision executes without an observed error. This one-step outcome is informative for intermediate execution states but is constant at some contact points. In particular, post-delivery execution produces no subsequent error-bearing step, and τ^2-telecom does not expose the same process-output channel used by SWE-bench and Terminal-Bench. For these cells, we instead use episode success as the outcome. The resulting estimates should therefore be interpreted accordingly: delivery-gate estimates remain closely tied to the final decision, whereas the τ^2 estimates reflect episode-level association under the training behavior distribution. The underlying evidence mapping is documented in Appendix A.2.

Results. Figure 9 shows the propensity-weighted effects for all estimable cells. The strongest and most consistent positive effects occur at delivery gates. On SWE-bench, consultation at DELIVER/CLEAN/$k = 0$ increases estimated episode success by 0.54 with a 95% bootstrap interval of $[0.15, 0.85]$. The corresponding effects after prior consultation are 0.24 $[0.04, 0.45]$ for DELIVER/CLEAN and 0.32 $[0.05, 0.55]$ for DELIVER/ERROR. Terminal-Bench shows the same pattern: DELIVER/CLEAN/$k \geq 1$ yields 0.33 $[0.12, 0.50]$ and DELIVER/ERROR/$k \geq 1$ yields 0.24 $[0.06, 0.44]$.

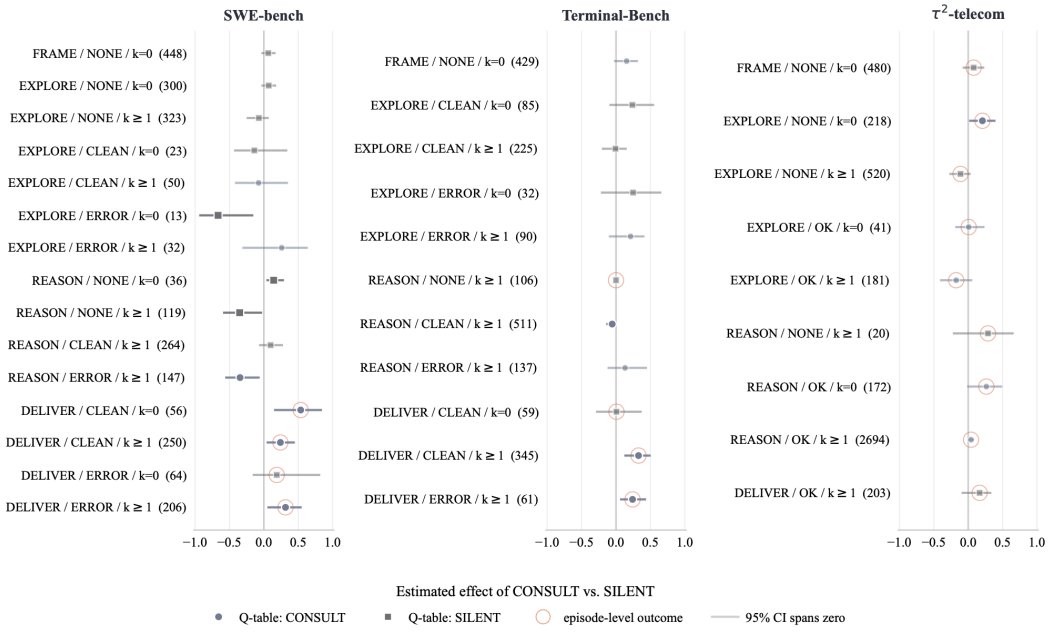

Figure 9: Propensity-weighted effect of CONSULT versus SILENT within each state cell. Points show Hájek IPW estimates with 95% episode-cluster bootstrap intervals. Marker shape indicates the greedy action of the reference Q-table; circled markers use the episode-level outcome because the one-step outcome is constant or unavailable. Faded intervals include zero. Parentheses report the number of observed decisions in each cell.

Mid-trajectory effects are more heterogeneous. On SWE-bench, REASON/NONE/$k \geq 1$ and REASON/ERROR/$k \geq 1$ have negative one-step estimates of -0.35 $[-0.59, -0.03]$ and -0.35 $[-0.56, -0.06]$, respectively. For the latter cell, however, the corresponding episode-level estimate is positive (0.23 $[-0.03, 0.46]$). This pattern is consistent with consultation causing additional intermediate work before any benefit appears in the final task outcome, illustrating the difference between immediate execution effects and the longer-horizon return optimized by ARIA.

Relation to the learned policy. Among the eleven estimated cells whose 95% interval excludes zero, the sign of the propensity-weighted effect agrees with the greedy action of the reference Q-table in eight; all five significant delivery-gate cells agree. The three disagreements occur in intermediate REASON states where the one-step estimate is negative while the learned policy selects CONSULT. This is not necessarily contradictory: the Q-table is trained on the remaining episodic return, whereas the primary IPW outcome measures the next execution step. The agreement at delivery gates is therefore the most direct comparison between the two estimands.

Limitations. The estimates use training trajectories because randomized exploration is required for propensity weighting; the frozen evaluation policy is greedy. They therefore characterize consultation effects under the observed training distribution rather than under the deployed policy. Cell occupancy is also highly uneven, and several cells are too sparse to estimate reliably. For τ^2-telecom, the analysis relies on episode success because the one-step execution outcome is unavailable. Finally, the reported intervals are not adjusted for multiple comparisons: 11 of 35 primary intervals exclude zero, so individual cells should be interpreted together with the broader cross-cell pattern rather than as isolated hypothesis tests.

C.6 MAST FAILURE ANALYSIS

Annotation protocol. We analyze failed evaluation trajectories using the 14-mode MAST taxonomy (Cemri et al., 2025), grouped into system design (FC1), inter-agent misalignment (FC2), and task verification (FC3). Each trajectory receives non-exclusive presence labels and one mutually exclusive primary failure mode. Labels are produced by `gemini-3.1-pro-preview` at high reasoning effort and temperature 0. Main-text comparisons use the primary label; presence labels are used only as supporting diagnostics.

Coverage. MAST is applied only to failed trajectories with a usable execution trace; successful trajectories are not judged. Episodes without a recoverable trace remain failures in the headline benchmark metrics but are excluded from MAST denominators. This affects Terminal-Bench in particular, where some crash-path episodes contain no trajectory suitable for annotation. Across benchmarks, over 1,600 failed trajectories are judged.

Human validation. Because the automatic judge belongs to the same model family as the advisor, we manually validate a stratified sample of 40 trajectories. We record the primary MAST mode and, for advisor conditions, separately determine whether relevant guidance was available to the executor and whether it was reflected in subsequent actions.

On the primary label, human annotation agrees with the automatic judge on 35 out of 40 trajectories (87.5%): 10/10 for both *Executor-only* and *Always*, 8/10 for *Advisor-only*, and 7/10 for ARIA. For context, MAST's original validation reports 94% accuracy and Cohen's $\kappa = 0.77$ between its LLM annotator and expert annotations, providing a reference point for the 87.5% exact primary-label agreement observed here (Cemri et al., 2025). The dominant errors are *ignored agent input* (FM-2.5) and *incorrect verification* (FM-3.3). Additionally, manual validation confirms that in every advisor trajectory whose final consult step flagged the run as off track, the relevant guidance had reached the executor, yet the executor made no corrective action and declared the task as satisfied. Human validation thus supports the claim that FM-2.5 reflects available advice that was not acted upon, rather than advice that failed to reach the executor.

Interpretation. MAST labels characterize the composition of residual failures and are not treated as causal attributions to individual consultations. Moreover, several FC2 modes require inter-agent communication to occur at all; comparisons of failure profiles should therefore be interpreted together with each condition's communication regime.

Secondary failure modes. The main-text analysis focuses on the mutually exclusive primary failure mode to characterize the dominant reason each unsuccessful trajectory breaks down. Figure 10 complements this view with all additional MAST modes observed in the same failed trajectories. Because these presence labels are non-exclusive, a single trajectory can contribute to multiple categories, revealing failure mechanisms that co-occur rather than determining only which one is judged primary.

The broader profiles reinforce the distinction highlighted in Section 5.4. On SWE-bench, verification-related failures remain common across conditions, but externally allocated consultation also introduces substantial inter-agent failure mass, particularly ignored agent input. Thus, improved consultation placement can remove one bottleneck while exposing another: relevant guidance may enter the trajectory without being incorporated by the executor. On Terminal-Bench, the secondary profiles remain more mixed, consistent with the finding that ARIA and *Heuristic* can achieve similar aggregate success while failing on different tasks and for different reasons. On τ^2-bench, inter-agent failure modes are comparatively less prominent, matching the much lower ignored-input rates discussed in the main text. The secondary labels therefore provide a more complete view of residual system behavior without changing the primary-failure comparisons reported in Section 5.4.

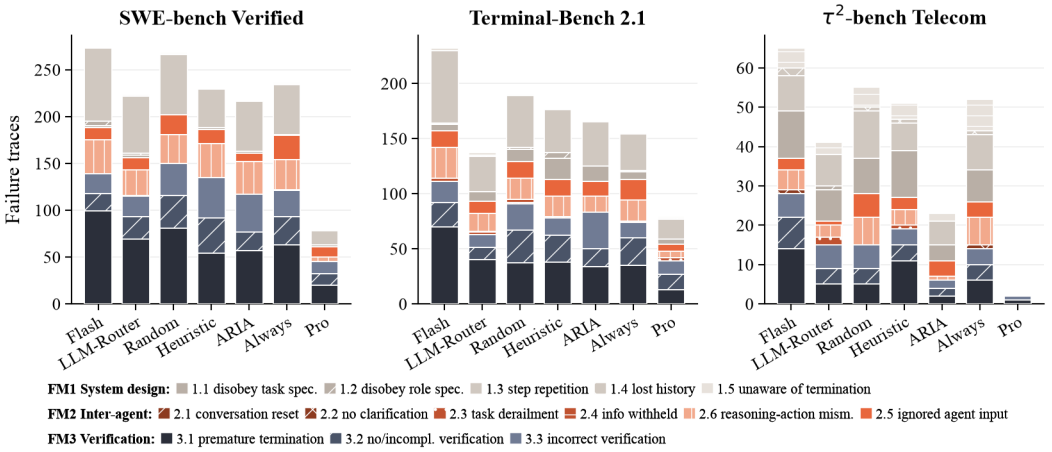

Figure 10: Secondary failure modes across baselines and benchmarks grouped into system-design, inter-agent misalignment, and verification failures, showing different residual failure profiles.