LEARNING WHAT TO SKIP: COUNTERFACTUAL CREDIT ASSIGNMENT FOR EFFICIENT MULTI-AGENT LLM WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent LLM workflows use planning, execution, verification, and summarization to improve task performance, yet the value of each component depends on the state already produced. Executing every component can waste computation or overwrite a correct intermediate answer. We formulate component omission as counterfactual credit assignment: full-workflow logs reveal the executed trajectory's reward, while controlled skip interventions reveal the consequences of omitting a future step. We introduce **Learning What to Skip** (LW2S), which learns action-specific safety models from these interventions and combines held-out calibration with domain-native guards to select skips. When an early skip is rejected, the controller can continue execution and reconsider a later component. Across mathematical reasoning, multiple-choice QA, and code generation with two instruction-model families, LW2S reduces recorded token cost while matching or improving aggregate full-workflow accuracy in the evaluated settings. Scale-up and second-topology experiments further examine component redundancy, while shared-error cases reveal why agreement alone is insufficient for skip selection. These findings connect efficient workflow execution to learning the conditional utility of individual components. Code is available at: https://anonymous.4open.science/r/LW2S-0DE7.

1 INTRODUCTION

Multi-agent LLM systems increasingly solve tasks by executing structured workflows rather than a single model call. A typical workflow may plan a solution, query multiple executors, verify intermediate answers, revise the result, and summarize a final response. This design has an intuitive appeal: more specialized computation can expose errors that one model call would miss. It is also costly. Each additional component consumes tokens and latency, and each extra reasoning step creates another opportunity for a plausible but wrong intermediate output to overwrite a correct one (Du et al., 2023; Wang et al., 2024; Zhuge et al., 2024; Yun et al., 2026; Tastan et al., 2026; Wu et al., 2026).

Recent agentic routing and graph-of-agents methods address part of this problem by deciding which agents to select or how agents should communicate (Wang et al., 2024; Du et al., 2023; Zhuge et al., 2024; Feng et al., 2026; Yun et al., 2026; Tastan et al., 2026). These methods shift multi-agent inference away from fully dense collaboration. However, they usually optimize workflow construction from end-to-end outcomes: which model should be called, which role should be active, or which graph should be generated for a query. They do not directly answer a finer counterfactual question: **after a workflow has already produced a partial state, which future components are still worth executing?**

Our starting observation is that component utility is not a role property. A verifier can repair an incorrect solution, leave a correct solution unchanged, or damage an answer by introducing a shared misconception. A summarizer can be useful when intermediate outputs are verbose or inconsistent, but redundant when the verifier has already produced executable code or a final answer. A planner may help on difficult tasks and be pure overhead on easy ones. The same component can therefore have positive, zero, or negative marginal utility across tasks and domains. This makes fixed pruning unsafe and full execution wasteful.

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.

We treat each optional workflow component as an intervention. Given a task x, an observed workflow prefix h_s, and a candidate future component a, we ask whether skipping a would preserve the reward of the full workflow while reducing cost. This is a counterfactual credit assignment problem for workflow computation. Proposition 1 formalizes the basic obstacle: full-workflow logs reveal the reward of the executed trajectory, but they do not identify the reward that would have been obtained by omitting a future component. The trace data must therefore contain controlled interventions such as omitting the verifier, omitting the summarizer, or bypassing an executor while keeping the rest of the workflow fixed.

We introduce **Learning What to Skip**, a selective decision framework for multi-agent LLM workflows. LW2S collects counterfactual traces by executing the full workflow and single-component skip interventions for the same task. It then trains action-specific safety models that estimate whether a skip action preserves full-workflow reward under the observed prefix. At deployment, LW2S accepts a skip only when held-out calibration and a domain-native guard support the action; otherwise it abstains to the full workflow or tries a later skip decision.

This formulation makes workflow computation conditional on component utility in the current state. Numerical agreement can support skipping in math reasoning, whereas option agreement in multiple-choice QA can reflect a shared misconception. In code generation, executable tests distinguish useful agreement from shared buggy logic. These differences connect the choice of skip action to domain-specific evidence, as organized in Appendix A.12.

We evaluate LW2S across mathematical reasoning, multiple-choice QA, and code generation with two instruction-model families, larger-model scale-up diagnostics, and a second debate-refine topology. Across the main evaluated settings, LW2S reduces component cost while matching or improving aggregate full-workflow accuracy. The MBPP slice-shift analysis shows how selecting between verifier and summarizer omissions recovers savings when early skipping fails. Appendix A provides the evaluation protocols, uncertainty estimates, and additional results.

Our contributions are:

- We formulate workflow efficiency as counterfactual credit assignment: full-workflow logs reveal executed rewards, but not the safety of skipping later components.
- We introduce LW2S, a calibrated selective controller with action-specific safety prediction, source-stable calibration, abstention, fall-through, and domain-native guards.
- We compare LW2S with RouteLLM, GraphPlanner, AutoMix, Prompt-LLM, and ablations under the same post-prefix skip interface across math, QA, and coding tasks.
- We provide model-generalization, scale-up, resource-accounting, second-topology, and failure-taxonomy evidence that separates safe redundancy from unsafe shared-consensus regimes.

2 Related Work

Structured multi-agent workflows and routing. Multi-agent LLM systems distribute reasoning across roles, model instances, and communication structures. Debate methods ask agents to critique and revise one another. Mixture-of-agents systems aggregate multiple proposals, and graph-based systems use selected agents, directed message passing, or learned workflow graphs to avoid dense all-to-all interaction (Du et al., 2023; Wang et al., 2024; Zhuge et al., 2024; Yun et al., 2026; Tastan et al., 2026; Wu et al., 2026). Routing methods make this efficiency objective more explicit: RouteLLM learns model-pair routing from preference data, while recent agentic routers extend routing to multi-step workflow generation and graph-structured memory (Ong et al., 2024; Feng et al., 2026; Yue et al., 2025). These methods decide which model, agent, or graph should be used before or during workflow construction. LW2S takes the workflow prefix as given and asks a later question: which future component is still worth running?

Selective computation and verification. Early-exit and selective-prediction methods abstain when prediction confidence is insufficient (Geifman & El-Yaniv, 2017; 2019; Xin et al., 2020). In LLM systems, related ideas appear as confidence-based routing, self-consistency stopping, consensus heuristics, and self/meta-verification cascades such as AutoMix (Aggarwal et al., 2024). Single-agent workflow compaction offers another route by simulating homogeneous multi-agent workflows

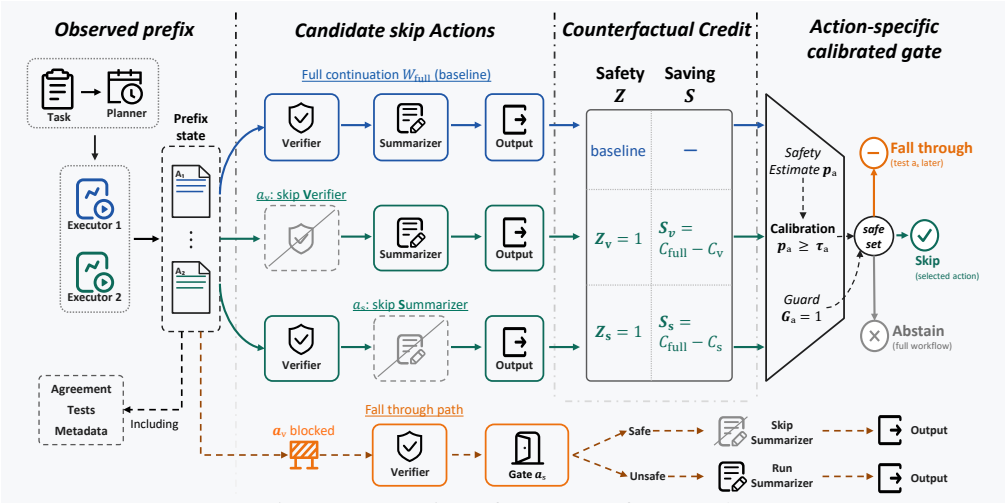

Figure 1: Overview of our LW2S framework.

inside one model and reusing context (Xu et al., 2026). These approaches exploit observable signals, but confidence, agreement, and compact execution can still conflate redundancy with correctness. Agreement is not correctness: executors can share an incorrect option on MMLU or produce structurally similar buggy code on MBPP. We therefore pair selective computation with calibrated counterfactual evidence and domain-native guards. In code domains, visible unit tests provide a useful checker signal, although verifier-scoring benchmarks show that small test suites can have limited discriminative power (Ficek et al., 2025).

Counterfactual credit for workflow components. Credit assignment usually attributes final reward to actions in a trajectory. Recent counterfactual credit methods for LLM collaboration restore fixed transcript prefixes to evaluate alternative agent decisions for policy optimization (Chen et al., 2026). Agentic workflows expose a more operational inference-time question: a future component is valuable only when executing it changes reward enough to justify its cost. Full trajectories cannot answer that question, because they never reveal what would have happened had a verifier, summarizer, or later turn been omitted. LW2S records single-component interventions and turns them into skip-safety labels. Component-level computation becomes a direct learning target, rather than an implicit consequence of end-to-end reward. Appendix A.3 summarizes the boundary between this intervention view, workflow routing, early exit, and workflow compaction.

3 Methodology

LW2S learns when to omit future workflow components from controlled counterfactual traces (Figure 1). It trains a separate controller over observable intermediate outputs, with recorded rewards and costs providing supervision. The base LLM remains frozen, and the controller operates through the workflow's text interface.

3.1 Workflow Setup

Let x denote a task and let a workflow $\mathcal{W} = (c_1, \ldots, c_K)$ consist of ordered components such as a planner, two executors, a verifier, and a summarizer. Executing component c_t produces an output o_t and updates the observable workflow state $h_t = (x, o_1, \ldots, o_t)$. The full workflow produces a final answer y^{full}, reward $R^{\text{full}} \in [0, 1]$, and cost C^{full}. In our experiments, the optimized cost is recorded component token cost,

$$C(y) = \sum_{c_t \in \text{exec}(y)} \big(\text{prompt_tokens}_t + \text{completion_tokens}_t\big),$$

summed over the components executed by trace y. Summed component latency is reported separately for resource accounting (Appendix A.1); threshold selection uses token cost.

A skip action $a \in \mathcal{A}_s$ is available at state h_s when it omits one or more future components while keeping the remaining workflow semantics fixed. For example, after both executors have produced candidate solutions, the action $\mathrm{skip_verifier}$ bypasses the verifier and passes executor outputs directly to the finalization path. After the verifier has run, $\mathrm{skip_summarizer}$ bypasses final summarization and emits the verified answer.

Definition 1 (Skip safety)**.** *For task x, state h_s, and candidate skip action a, let R^a and C^a be the reward and cost of the counterfactual workflow that takes action a. Action is reward-preserving if*

$$Z_a(x, h_s) = \mathbb{I}[R^a \geq R^{\mathrm{full}}] = 1. \tag{1}$$

Its realized saving is $S_a = C^{\mathrm{full}} - C^a$. The learning problem is to select a positive-saving action with $Z_a = 1$, or abstain to the full workflow when no action is reliable.

Skip safety is relative to the full workflow: preserving its reward can retain an incorrect answer when the full workflow itself fails. A skip can also improve reward by avoiding a downstream edit that damages a correct intermediate answer.

3.2 Counterfactual Trace Collection

For each task, we execute the full workflow and a set of single-component interventions. The resulting trace record contains the task metadata, component outputs, executed/skipped components, final answer, reward, total cost, and latency. This produces supervised examples of the form,

$$(x, h_s, a, Z_a, R^a, R^{\mathrm{full}}, C^a, C^{\mathrm{full}}).$$

The key design choice is controlled intervention: only the target component is removed, so reward and cost differences are attributable to that skip action under identical task and workflow conditions.

Counterfactual trace collection precedes controller training and calibration. At deployment, the controller selects a continuation from the observed prefix. Appendix A.4 specifies the intervention protocol for each skip action.

3.3 Action-Specific Safety Models

We train a safety model for each decision stage and skip action. For action a, model estimates

$$p_\theta(Z_a = 1 \mid x, h_s, a),$$

using text features from the task and observed component outputs, structured features extracted from intermediate answers, and task metadata. Numerical and option-letter agreement are available after the executors; verifier answer changes become available only after verification. For code tasks, executable features indicate whether the observed component outputs pass the visible MBPP unit tests. Features are constructed from the prefix available at the corresponding decision stage.

Each action has a distinct decision context. Verifier skipping depends on executor outputs, whereas summarizer skipping can use the verifier's response. Separate models associate these stage-specific features with the reward consequences of the corresponding omission.

Predictor implementation. Our implementation combines TF–IDF text features with standardized structured features in a class-balanced logistic regression model. The text branch uses lowercase unigrams and bigrams with a vocabulary capped at 1,500 features. The structured branch vectorizes prefix metadata and observable agreement or test signals. Training minimizes the regularized, class-weighted logistic loss for the binary reward-preservation label. When an action's training labels contain only one class, the implementation uses a constant predictor for that class. Each action model is fitted on training traces and then scored on calibration traces to select its acceptance threshold. Thus prediction estimates which omissions preserve reward, while calibration determines which score regions the controller will use.

3.4 Calibration and Abstention

Let $\mathcal{D}_{\mathrm{cal}}$ be a calibration set of counterfactual traces from held-out sources such as random seeds or task slices. For each action a, we sweep thresholds τ on the predicted safety probability. A threshold

is eligible only if it belongs to

$$\mathcal{T}_a = \left\{ \tau : \widehat{d}_a(\tau) \le d_{\max},\ \mathrm{UCB}_{1-\alpha}\Big(\widehat{d}_a(\tau), n_a(\tau)\Big) \le \rho,\ \widehat{R}_a(\tau) \ge \widehat{R}_{\mathrm{full}} \right\}, \tag{2}$$

where $\widehat{d}_a(\tau)$ counts selected calibration examples with $R^a < R^{\mathrm{full}}$, $n_a(\tau)$ is the selected count, and $\mathrm{UCB}_{1-\alpha}$ is a Wilson upper confidence bound on degradation risk, with α denoting the tail probability. The final threshold is chosen from $\mathcal{T}_a$ to maximize cost saving, with optional guard steps that move the threshold above unsafe regions of the calibration sweep. If $\mathcal{T}_a$ is empty, the action is excluded.

With multiple calibration sources, source-stable selection requires the acceptance conditions separately on every source. Equivalently, the eligible set is the intersection of the source-specific threshold sets. This retains source-level reward losses that pooling could obscure. Section 4.2 discusses the statistical interpretation, and Appendix A.9 reports the configuration and representative thresholds.

3.5 Domain-Native Guards

Calibration controls empirical skip risk, but the policy should also expose why a skip is plausible in the current state. We therefore define guard functions $G_a(x, h_s) \in \{0, 1\}$ that encode domain-native invariants. The policy may skip only when $G_a = 1$.

For math reasoning, a verifier-skip guard checks whether executors agree on the same final numeric answer. For MMLU, choice-letter consensus provides a stress test of agreement-based selection: shared misconceptions can satisfy this condition while degrading reward. For MBPP, the two-stage policy permits verifier skipping only when both executor outputs pass the visible unit tests. Its later summarizer decision uses the observed verifier state and the learned acceptance rule. Verifier test success is also evaluated as a standalone guard in Appendix A.13.

A failed guard excludes the corresponding action even when the learned model assigns it high safety probability. Guards thus constrain acceptance using explicit conditions on the available workflow state.

3.6 Selective Skip Policy

At a decision stage s, the policy forms a safe set

$$\mathcal{S}_s(x, h_s) = \{ a \in \mathcal{A}_s : p_\theta(Z_a = 1 \mid x, h_s, a) \ge \tau_a \ \wedge\ G_a(x, h_s) = 1 \}. \tag{3}$$

If $\mathcal{S}_s$ is empty, the workflow executes the next component. Otherwise, the policy selects the accepted action with the largest estimated saving:

$$\pi(x, h_s) = \operatorname*{arg\,max}_{a \in \mathcal{S}_s(x, h_s)} \widehat{S}_a(x, h_s). \tag{4}$$

For a sequential workflow, this decision can apply more than once. In the MBPP two-stage policy, the controller first tries to skip the verifier after the executors. If the code-execution guard blocks that action, the workflow runs the verifier, then tries to skip the summarizer. This fall-through behavior is central: the policy can recover efficiency later without taking an unsafe earlier shortcut.

4 Analysis of Counterfactual Skip Decisions

We analyze the information needed to learn skip decisions, the interpretation of empirical calibration, and the accounting of sequential fall-through.

4.1 Why Full-Workflow Logs Are Insufficient

The skip decision is a counterfactual question. At state h_s, the full log shows the reward after executing the remaining components, but it does not show what reward would have been obtained had a future component been omitted. This creates a basic identification problem for policies trained only on full-workflow trajectories.

Proposition 1 (Non-identifiability of skip safety from full logs)**.** *Fix a workflow prefix state h_s and a candidate skip action a. Suppose the available data contain only full-workflow observations $(x, h_s, R^{\text{full}}, C^{\text{full}})$, and suppose this distribution assigns positive probability to tasks with $R^{\text{full}} > 0$. Without an intervention record or a structural assumption linking R^a to R^{full}, the value of*

$$Z_a(x, h_s) = \mathbb{I}[R^a \geq R^{\text{full}}]$$

is not identifiable from the full-workflow distribution.

Proof. Consider any observed full-workflow distribution over $(x, h_s, R^{\text{full}}, C^{\text{full}})$. Construct two counterfactual worlds that share this exact distribution. In the first world, set the unobserved skip reward to $R^a = R^{\text{full}}$ for every task, so the skip action is reward-preserving. In the second world, keep the same full-workflow observations but set $R^a < R^{\text{full}}$ on any positive-mass subset of tasks where the omitted component is necessary. The observed full logs are identical in the two worlds, yet the skip-safety labels differ on that subset. No learner that observes only the full logs can distinguish these worlds. □

The proposition identifies a supervision gap: observing the full continuation alone leaves the reward of an omitted component undetermined. Paired skip traces supply this missing outcome for sampled tasks and prefixes. Generalizing these labels to new prefixes remains the task of the learned controller.

4.2 Interpretation of Empirical Calibration

Equation 2 defines an empirical acceptance certificate for the selected calibration region. The Wilson bound is evaluated for each candidate threshold, while the threshold itself is selected from a grid. These pointwise bounds do not provide simultaneous coverage over the threshold search or a guarantee under distribution shift. Held-out test degradation and uncertainty are therefore reported separately from calibration results (Appendix A.10).

Source-stable calibration constrains the eligible thresholds across observed sources; it does not prescribe a fixed test coverage. An accepted region can cover every task in a particular test slice. The MMLU row in Table 11, for example, reports 100 selected tasks out of 100. Coverage describes how often the policy takes an action, while task-level degradation measures its observed reward consequences.

4.3 Single-Intervention Accounting

At a fixed prefix and threshold, the guard in Equation 3 can only exclude candidate actions. Across a sequential workflow, however, rejecting an early action can expose a later decision. The relevant accounting unit is the complete selected continuation.

Proposition 2 (Fall-through recovery in a sequential policy)**.** *Consider a two-stage workflow with candidate actions a_1 at stage s_1 and a_2 at later stage s_2. If the policy attempts a_1 first and falls through to a_2 whenever a_1 is rejected or guard-blocked, then the set of selected actions is a subset of valid single-stage interventions $\{a_1, a_2, \text{full}\}$. The policy therefore never relies on an unobserved combined intervention, and any reward/cost assigned to a selected task corresponds to an actually collected counterfactual trace.*

For each task, reward and token cost are taken from the selected trace, including the verifier cost when execution reaches the later decision. Savings from separate interventions are never added. Appendix A.4 specifies the outputs and accounting for each continuation.

5 Experiments

5.1 Experimental Setup

Workflows. The main matrix uses a fixed five-role workflow with a planner, two executors, a verifier, and a summarizer. For every task, we save the full workflow and single-component skip interventions, and train only the skip controller; no base LLM weights are updated. Appendix A.2

Table 1: Main same-protocol evidence matrix. Acc. and Save are percentages; Acc. superscripts count task-level degradations against the full workflow. No superscript means zero observed degradation. Mistral MMLU averages three rotations.

Backbone	Setting	Metric	Full	RouteLLM	GraphPlanner	AutoMix	Prompt-LLM	LW2S
Qwen2.5-7B	MATH verifier	Acc.	65.2	64.8[1]	65.2[1]	65.6	64.8[1]	**66.4**
		Save	0.0	14.6	11.2	18.3	20.5	**25.1**
	MATH summarizer	Acc.	65.2	64.8[1]	65.2	65.2	64.8[1]	**65.2**
		Save	0.0	11.8	9.5	14.2	16.5	**18.4**
	GSM8K verifier	Acc.	88.5	88.0[1]	88.5	88.0[1]	88.0[1]	**88.5**
		Save	0.0	12.7	8.9	15.8	19.1	**20.6**
	MMLU verifier	Acc.	73.0	73.0[1]	73.0[1]	73.0[1]	72.0[1]	**73.0**[1]
		Save	0.0	5.8	4.3	7.0	7.7	**8.6**
	MBPP slice0	Acc.	69.0	68.0[1]	69.0	69.0	67.0[2]	**70.0**
		Save	0.0	16.9	13.4	22.1	24.7	**28.5**
	MBPP slice1	Acc.	63.0	62.0[1]	63.0	62.0[1]	61.0[2]	**63.0**
		Save	0.0	18.5	15.2	24.9	27.0	**31.2**
Mistral-7B	MATH verifier	Acc.	12.8	12.4[1]	12.8	12.6[1]	12.2[1]	**12.8**
		Save	0.0	11.3	8.8	14.6	17.0	**19.1**
	MATH summarizer	Acc.	12.8	12.8	13.0	13.2	12.4[1]	**14.0**
		Save	0.0	12.9	10.4	16.3	18.6	**21.8**
	GSM8K verifier	Acc.	39.5	38.7[1]	39.1[1]	39.0[1]	38.3[1]	**39.5**
		Save	0.0	13.6	10.7	17.5	20.1	**22.2**
	MMLU verifier	Acc.	55.0	54.6[1]	55.4	55.8	54.2[1]	**57.2**
		Save	0.0	15.7	12.1	19.2	22.4	**25.3**
	MBPP slice0	Acc.	28.0	28.0	29.0	30.0	27.0[1]	**32.0**
		Save	0.0	14.1	11.5	18.3	21.0	**23.7**
	MBPP slice1	Acc.	25.0	25.0	26.0	28.0	24.0[1]	**31.0**
		Save	0.0	14.5	11.8	18.6	21.4	**24.1**

describes the workflow-family scope, and Appendix A.7 reports a second debate-refine topology with different roles and information flow.

Benchmarks. We evaluate mathematical reasoning on MATH and GSM8K, multiple-choice QA on MMLU, and code generation on MBPP. These settings expose different skip signals: final-answer agreement in math, potentially misleading option agreement in QA, and executable unit-test evidence in code.

Models and resources. The main table uses Qwen2.5-7B-Instruct and Mistral-7B-Instruct, with Qwen2.5-14B-Instruct scale-up diagnostics in Appendix A.6. The saving metric is recorded component-token reduction relative to the full trace for the same task. Resource accounting, including trace counts, token totals, latency, hardware, and API use, is reported in Appendix A.1.

Baselines and policies. We compare against RouteLLM (Ong et al., 2024), GraphPlanner (Feng et al., 2026), AutoMix (Aggarwal et al., 2024), and Prompt-LLM under the same post-prefix skip protocol. Each policy observes the same workflow prefix and either accepts a skip action or falls back to the full workflow. Fixed skip, random skip, consensus-only, and guard-only variants serve as diagnostic ablations. Appendix A.5 gives the baseline specification and clarifies how routing and adaptive-computation principles are instantiated for component omission.

Comparison controls. All policies use the same saved workflow traces, data partition, reward parser, and token accounting. At a decision stage, inputs are restricted to the observed prefix. RouteLLM and GraphPlanner also receive intervention-derived binary skip-safety labels; LW2S organizes prediction and calibration by action and adds source checks, guards, and sequential fallthrough. Table 6 specifies the inputs, supervision, and deployment rule of each method.

5.2 Main Results

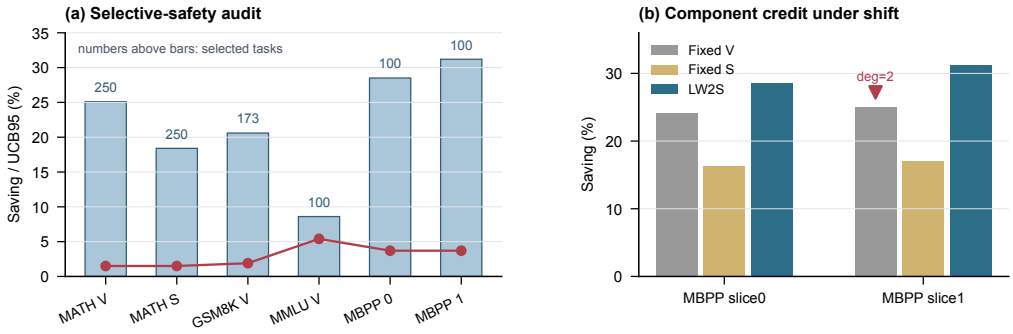

Figure 3: Diagnostics. (a) LW2S coverage, saving, and Wilson degradation-risk bounds on Qwen2.5-7B. (b) MBPP component credit under slice shift; LW2S reallocates verifier and summarizer skips while preserving reward.

LW2S reduces token cost in all six Qwen2.5-7B settings, with zero observed task-level degradations in five settings and one degradation on MMLU (Table 1). Under the shared skip protocol, RouteLLM, GraphPlanner, AutoMix, and Prompt-LLM also save tokens, and each incurs degradation in at least one setting. Figure 2 summarizes this trade-off across settings; Appendix A.8 provides the corresponding LW2S uncertainty estimates and full-workflow references.

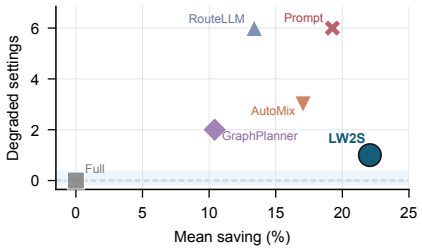

Figure 2: Qwen2.5-7B method summary; lower right is better.

On MATH, LW2S improves verifier-setting accuracy from 65.2% to 66.4% with 25.1% saving and omits summarization with 18.4% saving. It preserves full-workflow reward on GSM8K. On MMLU, aggregate accuracy remains 73.0% with 8.6% saving, although one task degrades. Figure 3(a) reports selected counts and Wilson degradation-risk bounds alongside these results. Appendix A.10 gives the corresponding sample sizes and uncertainty estimates.

The MBPP split is the main source-shift test. On Qwen2.5-7B slice0, LW2S improves pass rate from 69.0% to 70.0% while saving 28.5%. The strongest named-baseline saving reaches 24.7% but drops pass rate to 67.0%. On slice1, LW2S requires both learned acceptance and passing executor tests to skip the verifier. If either condition fails, it executes the verifier and evaluates the later summarizer skip. This preserves the full 63.0% pass rate while saving 31.2%, whereas the most aggressive Prompt-LLM baseline saves 27.0% but drops to 61.0%.

The same pattern survives a cross-family check with Mistral-7B. LW2S has zero observed degradations across all six settings, preserves or improves full-workflow reward, and improves the two MBPP slices by four and six pass-rate points. The competing rows are less stable: Prompt-LLM saves aggressively but degrades every setting, while AutoMix is stronger in several rows without reaching uniform zero-degradation behavior. Appendix A.6 adds a Qwen2.5-14B scale-up slice, where MATH and MBPP retain skip-safe redundancy but MMLU remains a high-saving unsafe boundary. The debate-refine diagnostic in Appendix A.7 checks topology shift; LW2S reaches 68.0% pass rate versus 67.0% for the full workflow while saving 16.65% recorded token cost.

5.3 Attributing the Gain to Component-Level Credit

Figure 3(b) examines component selection on MBPP. On slice1, fixed verifier skipping saves tokens but loses two pass-rate points. LW2S selects verifier skipping on 57 tasks and summarizer skipping after verification on the remaining selected tasks, preserving the full pass rate with the highest recorded saving. The comparison shows how access to a later omission recovers efficiency when early skipping fails; Appendix A.11 reports the baseline results and available action counts.

5.4 Code Generation as a Deterministic-Checker Domain

MBPP tests whether the framework transfers beyond math reasoning into a domain with executable correctness checks. On three initial 50-task seeds, summarizer skipping preserves full pass rate with zero degradations, while verifier skipping improves pass rate by three tasks in aggregate. Two disjoint unseen 100-task slices then separate safe verifier redundancy from unsafe shared-code errors.

The first slice supports verifier skipping: guarded verifier removal improves pass rate from 69% to 70% with zero degradations and 15.75% saving. The second slice exposes a boundary. Unguarded verifier skipping drops pass rate from 63% to 61%, with two degradations where both executors produced the same incorrect code and the verifier repaired it. The code-execution guard blocks these failures and restores the full pass rate, but saves only 12.89% when used as a single-stage verifier policy. On this slice, the two-stage LW2S policy preserves the same pass rate with 31.2% saving by combining accepted verifier omissions with later summarizer omissions. This comparison demonstrates the efficiency recovered through later decisions when early skipping is rejected.

5.5 What the Boundary Cases Show

The boundary cases identify the method's operating principle. On MMLU, option-letter agreement can reflect shared misconception. Appendix A.14 reports a case where both executors choose the same wrong option and the verifier repairs the answer. On MBPP, abstract-syntax equality between executor programs can reflect shared buggy logic. Appendix A.13 shows that AST equality alone selects 94 of 100 slice1 tasks and incurs two degradations, while executable evidence blocks those failures. Appendix A.12 organizes these cases together with redundant late summarization. In each regime, LW2S treats disagreement between calibration, guards, and observed state as a reason to abstain or fall through to a later decision.

5.6 Component Utility under Topology Shift

The debate-refine workflow replaces the main template's executor interaction with opposing code proposals followed by a critic and finalizer. On the 100-task MBPP slice, LW2S reaches 68.0% pass rate with 16.65% saving, compared with 67.0% for full execution. Fixed summarizer skipping reaches the same operating point, while fixed verifier skipping preserves 67.0% pass rate with 27.42% saving. All three have zero observed degradations (Table 8). These results identify a topology where a fixed late omission already captures the realized LW2S benefit. In contrast, the main MBPP slice1 requires retaining the verifier on part of the slice and recovering savings through later summarizer decisions. Together, the two workflows distinguish broadly redundant components from state-dependent omission opportunities.

6 Conclusion

We introduced LW2S, which learns when to omit future workflow components through counterfactual supervision, action-specific calibration, and domain-native guards. Controlled interventions expose the reward consequences of a skipped step, providing a learning signal absent from full-workflow logs alone. Across the evaluated reasoning and coding tasks, LW2S reduces token cost while matching or improving aggregate full-workflow accuracy. The component and topology analyses distinguish broadly redundant calls from omissions that depend on the current state. Shared-error cases show why agreement alone is insufficient, while sequential fall-through recovers savings when an earlier skip is rejected. These findings motivate learning the conditional utility of workflow components as a basis for adaptive multi-agent execution.

AI Use Statement

We used generative AI tools to assist with paper writing and proofreading. All claims, numerical results, scripts, and tables reported in this work were derived from the authors' local source files and experimental outputs. We take full responsibility for the final content of this work, including the text, statements, and code.

Reproducibility Statement

The experiments use open-weight instruction backbones, fixed random seeds, public benchmarks, and saved JSONL traces containing full and single-component counterfactual interventions. The repository records the trace schema, generation commands, calibration scripts, policy summaries, and resource footprints. We have released anonymized source code, trace-processing scripts, task identifiers, and generated summary tables as supplementary material.

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. AutoMix: Automatically mixing language models. In *Advances in Neural Information Processing Systems*, 2024.

Yanjun Chen, Yirong Sun, Hanlin Wang, Jinghan Wang, Xinming Zhang, Xiaoyu Shen, Wenjie Li, and Wei Zhang. Exact is easier: Credit assignment for cooperative LLM agents. *arXiv preprint arXiv:2603.06859*, 2026.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. GraphPlanner: Graph memory-augmented agentic routing for multi-agent llms. *International Conference on Learning Representations*, 2026.

Aleksander Ficek, Somshubra Majumdar, Vahid Noroozi, and Boris Ginsburg. Scoring verifiers: Evaluating synthetic verification in code and reasoning. *arXiv preprint arXiv:2502.13820*, 2025.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, 2017.

Yonatan Geifman and Ran El-Yaniv. SelectiveNet: A deep neural network with an integrated reject option. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2151–2159, 2019.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Nurbek Tastan, Samuel Horváth, and Karthik Nandakumar. Stochastic self-organization in multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 40242–40271, 2026.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024.

Tongtong Wu, Yanming Li, Ziye Tang, Chen Jiang, Linhao Luo, Guilin Qi, Shirui Pan, and Gholamreza Haffari. CARD: Towards conditional design of multi-agent topological structures. *International Conference on Learning Representations*, 2026.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, 2020.

Jiawei Xu, Arief Koesdwiady, Sisong Bei, Yan Han, Baixiang Huang, Dakuo Wang, Yutong Chen, Zheshen Wang, Peihao Wang, Pan Li, and Ying Ding. Rethinking the value of multi-agent workflow: A strong single agent baseline. *arXiv preprint arXiv:2601.12307*, 2026.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MASRouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

Sukwon Yun, Jie Peng, Pingzhi Li, Wendong Fan, Jie Chen, James Zou, Guohao Li, and Tianlong Chen. Graph-of-agents for multi-agent llm collaboration. *International Conference on Learning Representations*, 2026.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. *Proceedings of the 41st International Conference on Machine Learning*, pp. 62743–62767, 2024.

A Protocols, Diagnostics, and Additional Evidence

We detail trace collection, intervention protocols, baseline implementations, and calibration, followed by scale-up, topology, and failure analyses. Throughout, *accuracy* reports aggregate task reward and *Deg.* counts individual tasks on which a policy underperforms the full workflow. Improvements on some tasks can offset degradations on others, so we report both quantities alongside token savings.

A.1 Trace Schema, Splits, and Reproducibility

Trace schema. Each JSONL record corresponds to one benchmark task and contains the task identifier, benchmark split, prompt, gold target, full-workflow component outputs, single-component skip-intervention outputs, parsed rewards, component token counts, component latency, and structured state metadata. The metadata includes executor agreement, verifier decision fields, code-execution outcomes, and guard values. This design makes the unit of evidence a task-level counterfactual bundle rather than a single model response: for the same input, we can compare the full workflow against controlled omissions of future components.

Split discipline. Thresholds are selected on calibration traces before evaluation on a held-out trace source. With multiple calibration sources, a threshold must satisfy the degradation criterion separately on each source. If none qualifies, LW2S retains the full workflow or, in the MBPP two-stage setting, falls through to the later summarizer decision. Test intervention rewards are used to evaluate the selected policy.

Resource ledger. Table 2 reports the trace-generation resource ledger used by the paper. The table separates the main fixed-template matrix, the Mistral cross-family matrix, the Qwen2.5-14B scale-up diagnostics, and the second debate-refine workflow. This separation matters because the second workflow is used as a topology diagnostic rather than as part of the original main-matrix evidence.

Table 2: Resource accounting for reported open-weight traces. Five-role denotes planner, two executors, verifier, and summarizer; Debate denotes debate-refine. Latency is summed component latency and is not used for threshold selection; no commercial API calls are used.

Block	Backbone	Workflow	Tasks	Traces	Tokens (M)	Latency (h)	GPU	API
Main matrix	Qwen2.5-7B	Five-role	1100	6600	12.74	9.02	PRO6000	none
Model generalization	Mistral-7B	Five-role	1490	8940	20.31	14.52	PRO6000	none
Scale-up diagnostic	Qwen2.5-14B	Five-role	250	1500	3.27	4.62	PRO6000	none
Second workflow	Qwen2.5-7B	Debate	200	1200	3.14	2.90	PRO6000	none
Second workflow	Qwen2.5-7B	Debate	250	1500	3.83	2.97	PRO6000	none
Total	–	–	3290	19740	43.30	34.04	PRO6000	none

The accounting unit is a saved full/intervention trace. Token cost is computed from recorded prompt and completion tokens for the components executed in that trace. Reported latency is summed component latency from the same records. It is included to make the computational footprint auditable, but it is not used to tune thresholds or select policies. We do not reconstruct end-to-end wall-clock time from these logs because runs can be interrupted, resumed, and scheduled differently; using recorded component latency gives a stable per-trace resource diagnostic. All reported traces use open-weight models on one NVIDIA RTX PRO 6000 Blackwell Server Edition, and no commercial API calls are used.

Table 3: Workflow-family coverage. The main matrix uses a controlled five-role template; scale-up and debate-refine rows test persistence beyond one backbone and one topology.

Workflow family	Benchmarks	Backbones	Evidence role
Five-role	MATH, GSM8K, MMLU, MBPP	Qwen2.5-7B, Mistral-7B	Main safety–efficiency matrix with same-protocol named baselines and calibrated skip policies.
Scale-up fixed template	MATH, MMLU, MBPP	Qwen2.5-14B	Larger-backbone diagnostic for skip-safe redundancy and unsafe high-saving shortcuts.
Debate-refine	MBPP	Qwen2.5-7B	Topology diagnostic with different roles and information flow from the main workflow.

Table 4: Technical boundary between LW2S and adjacent efficiency paradigms. The key distinction is component-level skip supervision from controlled interventions after a workflow prefix has been observed.

Method family	Decision target	Supervision or evidence	Boundary to LW2S
Model routers	choose a model before execution	preference or task reward	Optimize model/cost trade-off; no component-level skip intervention labels or sequential fall-through.
Graph/workflow routers	choose a graph or agent allocation	end-to-end reward	Estimate workflow utility; usually does not identify whether a specific future component is safely omissible.
Selective prediction and early exit	emit, reject, or exit at a confidence gate	labels plus reject loss	Controls prediction risk–coverage; does not intervene on verifier/summarizer components in an observed workflow prefix.
Counterfactual MARL credit	assign action advantage during policy learning	counterfactual returns	Uses counterfactual credit, but it is not a calibrated deployment-time skip policy over LLM workflow components.
Workflow compaction	replace an evaluated workflow before deployment	workflow search or evaluation	Finds fewer turns or a single-agent simulation; no per-task fall-through after a blocked component skip.
LW2S	omit a specific future component after observing a prefix	full and single-skip traces	Learns action-specific skip labels, calibrates source-stable risk, and can fall through to later skip decisions.

A.2 Workflow-Family Coverage

The main matrix fixes the planner–executor–executor–verifier–summarizer template to make intervention labels comparable across benchmarks. The debate-refine workflow changes agent roles and information flow, testing whether component-level skip opportunities persist under a different topology. Table 3 lists the templates, backbones, and tasks covered by these evaluations.

A.3 Boundary to Adjacent Paradigms

Table 4 positions LW2S relative to routing, early-exit, counterfactual-credit, and workflow-compaction methods. The key distinction is the decision target. A model router selects which model to call; a workflow router chooses a graph or agent allocation; an early-exit method rejects or emits a prediction based on confidence. LW2S instead observes a partially executed multi-agent workflow and asks whether a specific future component is still needed for that task.

Table 4 compares the decision targets and supervision of these paradigms. LW2S learns the reward consequences of individual omissions. Its sequential policy (Section 3.6) can retain an early component and reconsider a later one, allowing the omission to depend on the evolving workflow state.

A.4 Intervention Protocol

The intervention protocol in Table 5 fixes the workflow template, prompts, decoding settings, answer parsers, and task family while changing exactly one future component. The observed prefix is the state available to the controller at test time. For verifier skipping, the controller has seen both executor outputs but has not run the verifier. For summarizer skipping, it has seen the verifier output and decides whether the summarizer is still needed. This design prevents leakage from components that would not have been available at the decision point.

Table 5: Controlled intervention protocol. Prompts, decoding settings, parsers, and workflow templates are fixed within each task family. Each selected continuation omits a single target component.

Action	Observed prefix	Removed component	Emitted/evaluated output	Accounting
skip verifier	after both executors	verifier	summarizer or finalizer consumes executor answers; the domain parser scores the final answer	token cost; latency logged separately
skip summarizer	after verifier	summarizer	verified answer is emitted directly and parsed by the same task parser	token cost; latency logged separately
two-stage MBPP	after executors, then after verifier if blocked	one component per task	selected recorded intervention: skip verifier, skip summarizer, or full workflow; unit tests score the result	token cost of selected trace; latency logged separately

Table 6: Baseline inputs, supervision, and decision rules under the shared component-omission protocol. Methods use the same saved traces and data partition.

Method	Input state	Training signal	Calibration data	Deployment rule
RouteLLM	observed prefix features	binary skip-safe label	same split	threshold sweep; fallback to full workflow
GraphPlanner	prefix plus memory features	binary skip-safe label	same split	threshold sweep; fallback to full workflow
AutoMix	prefix confidence and action value	trace reward and cost	same split	value threshold; fallback to full workflow
Prompt-LLM	task, prefix, candidate action	in-context skip judgment	calibration split	probability threshold; fallback to full workflow
LW2S	prefix, agreement, metadata	action-specific intervention label	same split plus source checks	source-stable threshold with guard; fallback to full or later skip

The MBPP two-stage policy applies the learned threshold and code guard jointly at the verifier decision. Rejection by either condition leads to verifier execution and a later summarizer decision. Each task is assigned the reward and token cost of exactly one recorded continuation: skip verifier, skip summarizer, or full workflow. Accordingly, the number satisfying a standalone guard and the number selected by LW2S measure different predicates, even on identical tasks and traces.

A.5 Baseline Evaluation Protocol

The named baselines share LW2S's saved workflow traces, train/calibration/test partition, reward parser, and cost accounting. At test time, each decision rule receives the observed prefix and its available tool outputs; intervention rewards remain evaluation targets. This protocol holds workflow execution fixed across decision rules.

Table 6 specifies the implementations evaluated through the shared component-omission interface. RouteLLM (Ong et al., 2024) uses observed prefix features for a binary skip decision, and GraphPlanner (Feng et al., 2026) augments the prefix with memory features. Both receive binary skip-safety supervision derived from the intervention traces. AutoMix (Aggarwal et al., 2024) uses trace reward and cost for cost-sensitive selection, while Prompt-LLM asks the backbone to judge a candidate skip in context. LW2S organizes prediction and calibration by action and combines source checks with guards and sequential fall-through. Thus the comparison varies the decision rule and its use of supervision; intervention-derived labels are shared by several methods. Fixed skip, random skip, consensus-only, and guard-only variants serve as diagnostic ablations.

A.6 Qwen2.5-14B Scale-Up Diagnostics

The Qwen2.5-14B experiment evaluates fixed and consensus-based omissions on MATH seed55, MMLU seed2, and MBPP slice1 (Table 7). These diagnostic policies test how component utility changes with backbone scale. Verifier and summarizer redundancy persists on MATH, and MBPP retains code-generation skip opportunities. On MBPP, however, summarizer removal incurs an individual degradation despite increasing aggregate pass rate.

On MMLU, fixed verifier skipping raises accuracy from 82.0% to 83.0%, and executor-consensus skipping raises it to 84.0%. Both also cause individual degradations: gains on other tasks offset

Table 7: Qwen2.5-14B scale-up diagnostics. Rows test whether skip-safe and boundary phenomena persist under a larger backbone; Save is token-cost reduction.

Setting	Policy	Full	Acc.	Deg.	Save	Interpretation
MATH seed55	fixed skip verifier	66.0	70.0	0	25.0	verifier removal improved reward on this evaluated slice
MATH seed55	executor-consensus verifier skip	66.0	68.0	0	13.5	guarded redundancy remained after scaling the backbone
MATH seed55	fixed skip summarizer	66.0	66.0	0	17.2	summarizer removal preserved full-workflow reward
MMLU seed2	fixed skip verifier	82.0	83.0	3	23.6	boundary case: high saving did not imply safe skipping
MMLU seed2	executor-consensus verifier skip	82.0	84.0	2	22.0	consensus did not remove the shared-error failure mode
MBPP slice1	fixed skip verifier	69.0	70.0	0	24.5	verifier redundancy persisted on the scaled coding slice
MBPP slice1	fixed skip summarizer	69.0	72.0	1	17.0	summarizer removal was not uniformly safe at this scale

Table 8: MBPP debate-refine diagnostic on Qwen2.5-7B. Acc. is pass rate, Deg. counts task-level degradations, and Save is token-cost reduction.

Policy	Full	Acc.	Deg.	Save	Interpretation
Full workflow	67.0	67.0	0	0.00	reference execution of the debate-refine workflow
LW2S after executors	67.0	68.0	0	16.65	learned verifier/summarizer skip decision after observing proposer and opponent outputs
LW2S after verifier	67.0	68.0	0	16.65	learned summarizer skip decision after the critic/verifier state is available
Hindsight reward-improving skip	67.0	69.0	0	25.79	hindsight diagnostic, not a deployed policy
Fixed skip verifier	67.0	67.0	0	27.42	high-saving fixed intervention that is safe on this evaluated slice
Fixed skip summarizer	67.0	68.0	0	16.65	fixed late-component omission in the debate-refine workflow
Executor-consensus skip verifier	67.0	68.0	0	16.30	guard-only early skip based on proposer/opponent agreement
Code-guard skip verifier	67.0	68.0	0	15.42	guard-only early skip using executable-code evidence

the loss of answers repaired by the verifier. Shared-error failures therefore persist with the larger backbone, even where average accuracy and token savings improve.

A.7 Second-Workflow Debate-Refine Diagnostic

To examine component utility under topology shift, we evaluate a debate-refine workflow on MBPP. A framer initializes the problem, two agents produce opposing code proposals, a critic/verifier inspects the candidates, and a finalizer emits the answer. This changes the agent roles and information flow while retaining the verifier and summarizer intervention interfaces used by LW2S.

Section 5.6 compares the resulting LW2S operating point with fixed verifier and summarizer omissions. Table 8 additionally reports the two decision stages, hindsight selection, and consensus- and code-guard policies.

Table 8 reports the parsed result for this second workflow. The full debate-refine workflow reaches 67.0% pass rate on the 100-task MBPP slice. The LW2S rows improve aggregate pass rate to 68.0% while saving 16.65% of token cost. The hindsight diagnostic indicates that additional reward-improving omissions exist in the recorded interventions, while fixed and guard-only rows show that this evaluated slice also contains simple safe omissions. All listed rows have zero observed task-level degradations on this evaluated slice. The result serves a specific generalization purpose: the skip-credit signal is not confined to the original fixed-template workflow.

A.8 Unified Safety-Efficiency Ledger

Table 9 consolidates LW2S accuracy, task-level degradation, Wilson upper bounds, and token savings across backbones and workflows. The full-workflow rows provide the accuracy and cost references for each comparison.

The Qwen2.5-7B entries complement the method-level comparison in Figure 2. The additional rows report cross-family and topology results under the same accuracy, degradation, and cost definitions.

Table 9: Unified safety-efficiency ledger for headline LW2S evidence. UCB_{95} is a Wilson upper bound on observed task-level degradation for the evaluated slice; Save is token-cost reduction.

Setting	Backbone	Workflow	Policy	Full	Acc.	Deg.	UCB_{95}	Save
MATH verifier	Mistral-7B	Five-role	LW2S transfer	12.8	12.8	0	1.5	19.1
MATH verifier	Qwen2.5-7B	Five-role	LW2S final	65.2	66.4	0	1.5	25.1
MATH verifier	Qwen2.5-7B	Five-role	Full workflow	65.2	65.2	0	1.5	0.0
MATH summarizer	Mistral-7B	Five-role	LW2S transfer	12.8	14.0	0	1.5	21.8
MATH summarizer	Qwen2.5-7B	Five-role	LW2S final	65.2	65.2	0	1.5	18.4
MATH summarizer	Qwen2.5-7B	Five-role	Full workflow	65.2	65.2	0	1.5	0.0
GSM8K verifier	Mistral-7B	Five-role	LW2S transfer	39.5	39.5	0	1.9	22.2
GSM8K verifier	Qwen2.5-7B	Five-role	LW2S final	88.5	88.5	0	1.9	20.6
GSM8K verifier	Qwen2.5-7B	Five-role	Full workflow	88.5	88.5	0	1.9	0.0
MMLU verifier	Mistral-7B	Five-role	LW2S transfer	55.0	57.2	0	1.3	25.3
MMLU verifier	Qwen2.5-7B	Five-role	LW2S final	73.0	73.0	1	5.4	8.6
MMLU verifier	Qwen2.5-7B	Five-role	Full workflow	73.0	73.0	0	3.7	0.0
MBPP slice0 two-stage	Mistral-7B	Five-role	LW2S transfer	28.0	32.0	0	3.7	23.7
MBPP slice0 two-stage	Qwen2.5-7B	Five-role	LW2S final	69.0	70.0	0	3.7	28.5
MBPP slice0 two-stage	Qwen2.5-7B	Five-role	Full workflow	69.0	69.0	0	3.7	0.0
MBPP slice1 two-stage	Mistral-7B	Five-role	LW2S transfer	25.0	31.0	0	3.7	24.1
MBPP slice1 two-stage	Qwen2.5-7B	Five-role	LW2S final	63.0	63.0	0	3.7	31.2
MBPP slice1 two-stage	Qwen2.5-7B	Five-role	Full workflow	63.0	63.0	0	3.7	0.0
MBPP debate-refine	Qwen2.5-7B	Debate	LW2S after exec.	67.0	68.0	0	3.7	16.7
MBPP debate-refine	Qwen2.5-7B	Debate	LW2S after verif.	67.0	68.0	0	3.7	16.7
MBPP debate-refine	Qwen2.5-7B	Debate	Full workflow	67.0	67.0	0	3.7	0.0

Table 10: Representative LW2S calibration results. UCB_{80} is the Wilson upper bound used by the acceptance rule ($\alpha = 0.2$); saving is measured on the calibration traces.

Setting	cal. n	τ	Deg.	UCB_{80}	Cal. saving
MATH summarizer	140	0.95	0	0.5	10.5
GSM8K verifier	50	0.00	0	1.4	26.2
MBPP verifier	150	1.00	0	0.5	24.5
MBPP summarizer	150	0.00	0	0.5	16.6

Table 11: Coverage and uncertainty for the Qwen2.5-7B LW2S results in Table 1. Selected counts tasks assigned a skip; UCB_{95} bounds degradation over all evaluated tasks.

Setting	n	Selected	Deg.	UCB_{95}	Save
MATH verifier	250	250	0	1.5	25.1
MATH summarizer	250	250	0	1.5	18.4
GSM8K verifier	200	173	0	1.9	20.6
MMLU verifier	100	100	1	5.4	8.6
MBPP slice0	100	100	0	3.7	28.5
MBPP slice1	100	100	0	3.7	31.2

A.9 Calibration Certificates and Uncertainty

Calibration follows Equation 2, with $\alpha = 0.2$, $d_{\max} = 0$, $\rho = 0.15$, and a threshold grid in increments of 0.05. Source-stable variants apply the acceptance conditions separately to each calibration source. Table 10 reports representative accepted regions. Section 4.2 explains the statistical interpretation of this threshold search.

The columns list calibration size, selected threshold, observed degradation, UCB_{80}, and calibration saving. Threshold selection uses these calibration measurements before held-out test evaluation.

Table 11 complements the calibration view with test-set coverage and uncertainty diagnostics. The selected column counts how many tasks were routed away from the full workflow by LW2S. Five settings have zero observed degradations, for which the Wilson upper bound depends on sample size. MMLU has one degradation among 100 tasks, yielding a 5.4% upper bound, compared with 3.7% for the zero-degradation 100-task MBPP slices.

A.10 LW2S Coverage and Degradation-Risk Diagnostics

Coverage and saving measure different aspects of the policy. Coverage counts tasks assigned a skip, whereas saving also depends on the token cost of the omitted continuation. Thus similar selected counts can yield different savings across settings. The MMLU entry reports full test coverage (100/100); source-wise calibration does not imply low coverage on this test slice. The degradation count and Wilson bound describe the corresponding observed reward losses and finite-sample uncertainty.

Table 12: MBPP component-credit diagnostic. n_v and n_s count verifier- and summarizer-skip selections; – denotes unreported counts. Baseline accuracy, degradation, and saving match Table 1.

Slice	Method	Full	Acc.	Deg.	n_v	n_s	Save
MBPP slice0	RouteLLM	69.0	68.0	1	–	–	16.9
	GraphPlanner	69.0	69.0	0	–	–	13.4
	AutoMix	69.0	69.0	0	–	–	22.1
	Fixed skip verifier	69.0	70.0	0	100	0	24.2
	Fixed skip summarizer	69.0	69.0	0	0	100	16.3
	LW2S (ours)	69.0	70.0	0	69	31	**28.5**
MBPP slice1	RouteLLM	63.0	62.0	1	–	–	18.5
	GraphPlanner	63.0	63.0	0	–	–	15.2
	AutoMix	63.0	62.0	1	–	–	24.9
	Fixed skip verifier	63.0	61.0	2	100	0	25.1
	Fixed skip summarizer	63.0	63.0	0	0	100	17.0
	LW2S (ours)	63.0	63.0	0	57	43	**31.2**

Table 13: Failure and abstention taxonomy. The same visible prefix can indicate safe redundancy in one domain and an unsafe shared error in another.

Regime	Observable state	Mechanism	Evidence	LW2S response
Shared wrong consensus	Executors agree on the same answer, but the verifier changes it to the gold answer	Agreement is not a certificate when agents share a misconception or shortcut	MMLU verifier boundary; Table 15; Qwen2.5-14B MMLU rows	Accept calibrated actions; retain the verifier when acceptance fails
Superficial code agreement	Programs look similar, but visible tests expose a shared bug	Non-executable agreement can preserve the same incorrect logic across executors	MBPP slice1 AST-guard degradation in Table 14	Use executable guards and fall through to later decisions when early removal is blocked
Redundant late summarization	A verifier or checked intermediate already contains a parsable final answer or executable code	The final summarizer can add cost without improving reward, and can overwrite a correct intermediate answer	MATH summarizer, MBPP two-stage, and debate-refine rows	Prefer late-component omission when calibrated evidence supports reward preservation

A.11 Component-Level Credit Diagnostic

Figure 3(b) and Table 12 report the MBPP component-selection diagnostic. The LW2S rows match Table 1: slice0 selects 69 verifier skips and 31 summarizer skips, with 70.0% accuracy and 28.5% saving; slice1 selects 57 and 43, respectively, with 63.0% accuracy and 31.2% saving. Both have zero observed task-level degradations. These action counts show how the policy allocates omissions across the two decision stages.

A.12 Failure and Abstention Taxonomy

Table 13 groups the boundary cases into three mechanism-level regimes: shared wrong consensus, superficial code agreement, and redundant late summarization. The taxonomy explains why LW2S sometimes skips aggressively, sometimes abstains, and sometimes delays the skip decision to a later workflow state.

The first two regimes expose failures of agreement-based selection. MMLU executors can share a wrong option, and MBPP executors can share a bug despite AST-level similarity. Redundant late summarization provides a complementary opportunity: a verifier or checked intermediate output may already contain the final answer. Together, these regimes explain why component utility depends on the observed prefix and why a blocked early omission can leave a useful later skip available.

A.13 Guard Ablations

Table 14 compares prefix-available guard conditions and an offline structural diagnostic on two unseen 100-task MBPP slices. Executor test outcomes and executor AST equality are available before verification; verifier test outcomes become available after verification. Verifier/summarizer AST equality requires the completed summarizer output and is evaluated in hindsight to characterize unchanged code. It is excluded from online summarizer-skip decisions.

AST equality is the clearest failure mode. On slice0 it appears safe, selecting 93 tasks with no degradation. On slice1 it selects 94 tasks but incurs two degradations, because both executors can

Table 14: MBPP guard and structural diagnostics on unseen 100-task slices. Verifier/summarizer AST equality is a hindsight diagnostic; all other conditions use outputs available at their decision stage.

Slice	Condition	Full	Acc.	Deg.	Selected
slice0	both executors pass	69.0	70.0	0	69
	Executor AST equality	69.0	70.0	0	93
	both pass + AST	69.0	70.0	0	65
	verifier passes tests	69.0	69.0	0	69
	verifier/summarizer AST (hindsight)	69.0	69.0	0	99
slice1	both executors pass	63.0	63.0	0	58
	Executor AST equality	63.0	61.0	2	94
	both pass + AST	63.0	63.0	0	58
	verifier passes tests	63.0	63.0	0	63
	verifier/summarizer AST (hindsight)	63.0	63.0	0	100

Table 15: MMLU verifier-skip degradation case. Skipping retains the executors' shared wrong option A, while verification corrects it to the gold option C.

Example	Subject	Short prompt cue	Failure signal	Skip	Verifier	Gold
mmlu_test_13860	virology	What is a virus pandemic?	executor consensus A/A	A	C	C

share the same syntactic or structural bug. Guards that require executable evidence, such as both executors passing visible tests, are more conservative but block those slice1 failures. This supports the main design principle: guards should encode domain-native invariants rather than superficial agreement signals.

A.14 MMLU Failure Case

Table 15 reports the MMLU failure case: both executors select option A, while the verifier corrects the answer to C. Skipping verification retains the shared error.

Answer agreement is compatible with both redundant verification and a shared error that verification repairs. Intervention labels distinguish these outcomes during training, while source-specific calibration determines which predicted skip regions are accepted. This case illustrates the feature ambiguity behind agreement-based selection and the observed MMLU degradation despite unchanged aggregate accuracy.