DYNAMAS: ONLINE-ADAPTIVE DYNAMIC WORKFLOW ROUTING FOR MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly deployed in multi-agent systems, where specialized agents coordinate to perform planning, generation, and refinement. However, most existing multi-agent workflows are static: the same sequence of agents is applied to every query regardless of its difficulty, ambiguity, or evidence requirements. This leads to inefficient over-provisioning on simple queries and insufficient reasoning or validation on harder ones. To address this inefficiency, we propose DynaMAS, a training-free dynamic workflow routing framework for multi-agent systems. DynaMAS adapts inference-time computation to each instance by observing intermediate outputs and deciding whether to stop, continue, or invoke additional corrective steps. The framework uses a two-tier cascade: a lightweight Scout performs low-cost checks at each workflow step, while a stronger LLM-based Judge is invoked only for ambiguous or high-risk cases. Evaluation shows that DynaMAS reduces output-token consumption by 46.1% and end-to-end latency by 47.8% on average, while preserving or improving task performance across benchmarks.

1 INTRODUCTION

Large language models (LLMs) increasingly power *multi-agent systems* (MAS), in which specialized agents—planners, solvers, verifiers, critics, and repair agents—are composed into workflows. Such systems achieve strong results on code generation Wu et al. (2023), tool-augmented reasoning Li et al. (2023), data analysis, and information retrieval Yao et al. (2022); Qin et al. (2024); Wang et al. (2024). Deliberative patterns including chain-of-thought Wei et al. (2022), self-consistency Wang et al. (2022), self-refinement Madaan et al. (2023), and multi-agent debate Du et al. (2023); Shinn et al. (2023); Feng et al. (2024); Zhou & Zanette (2024) further improve robustness on hard reasoning tasks. Despite this progress, most multi-agent systems still rely on *fixed* workflows. The pipeline is either hand-designed or discovered by automated frameworks such as ADAS Hu et al. (2025) and AFlow Zhang et al. (2025b), but at runtime every query follows the same script of generation, verification, testing, and repair. This design assumes that a single workflow structure can serve a broad range of tasks, difficulty levels, and risk profiles. However, in practice, user queries differ widely in complexity, ambiguity, evidence requirements, safety sensitivity, and tolerance for error. Thus, a workflow tuned for one query is often wasteful for another and insufficient for a third.

This mismatch produces two complementary failure modes. *Over-provisioning* occurs when simple queries are routed through long refinement pipelines that add latency and cost with little or no improvement to the answer. *Under-provisioning* occurs when complex or ambiguous queries lack the multi-step reasoning or validation they actually need, leading to avoidable errors. As we show in section 3, on a Retrieval-Augmented Generation (RAG) benchmark, a searched workflow is over-provisioned on roughly 30% of requests and under-provisioned on 23%; only 6% are well matched to the predefined pipeline. The situation parallels adaptive computation in deep networks Teerapittayanon et al. (2016); Kaya et al. (2019); Graves (2016), where early exiting and conditional computation reduce cost by matching depth to instance difficulty.

These observations motivate a simple principle: **multi-agent workflows should be dynamic**. Rather than treating every query as equally hard, the system should adapt the number and type of steps to each instance. Easy queries should exit after one or two steps; harder ones should trigger additional verification or reasoning steps. This requires an explicit *router* that observes intermediate artifacts

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.

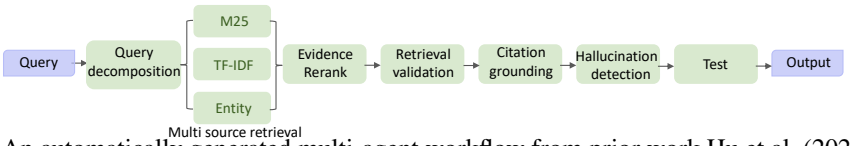


Figure 1: An automatically generated multi-agent workflow from prior work Hu et al. (2025); Zhang et al. (2025b). Multiple refinement, validation, and verification stages can improve accuracy on hard instances, but they add substantial latency and computational overhead when applied uniformly to every query.

and decides how to advance each query through the workflow. Crucially, this decision cannot be made only once at the start, as in prior MAS routers Zhang et al. (2025a); Yue et al. (2025): the need for further computation often becomes apparent only after intermediate outputs are produced, and initial-query routing ignores the variability of agent behavior mid-workflow.

To address this mismatch, we propose **DynaMAS**, a cascade **Dyna**mic workflow routing framework for **M**ulti-**A**gent **S**ystems. DynaMAS replaces static, one-size-fits-all pipelines with a router that selects the next computation based on the current intermediate output. It acts as a policy layer over existing workflows: it observes intermediate outputs and decides whether to early-exit, continue, or augment the output with an additional enhancement step. The design follows a two-tier cascade. The first tier is a low-overhead *Scout* that combines rule-based checks with a small language model; it runs at every step and handles clear low- or high-confidence cases at near-zero cost. The second tier is an LLM-based *Judge*, invoked only when the Scout flags an ambiguous or borderline case, that makes routing decisions over a compact, structured view of the workflow state.

Across four benchmarks, DynaMAS reduces token consumption by 46.1% and end-to-end latency by 47.8% on average, with no accuracy drop compared to the baseline workflow.

Contributions.

1. We identify a key limitation of current multi-agent systems: heavy, static pipelines apply the same steps to every query, leaving most workflows either *over-provisioned* (wasting compute on easy cases) or *under-provisioned* (failing on hard ones).
2. We propose a *two-tier routing system* that pairs low-cost rule-based checks with LLM-based decision-making to route workflows efficiently. A lightweight *Scout* runs at every step at near-zero cost; a larger LLM-based *Judge* is invoked only on the ambiguous cases the Scout flags, providing more precise routing decisions.
3. We introduce a *training-free online* design in which a summarizer LLM proposes routing options on the fly, allowing DynaMAS to be applied to new workflows without retraining or transfer overhead.

2 Related Work

2.1 Agentic Workflows

LLMs are increasingly used in structured agentic workflows rather than as single-step generators. Prior work improves reasoning through intermediate steps, search, feedback, and refinement, including Chain-of-Thought Wei et al. (2022), self-consistency Wang et al. (2022), ReAct Yao et al. (2022), Tree of Thoughts/Graph of Thoughts Yao et al. (2023); Besta et al. (2024), planning Hao et al. (2023); Wu et al. (2024a); Qiao et al. (2024b); Nayak et al. (2024), tree search Zhou et al. (2025), refinement Madaan et al. (2023), reflection Shinn et al. (2023), tool use Schick et al. (2023), and parallel execution Kim et al. (2023). These methods show that decomposing a task into multiple reasoning or tool-use steps can improve final performance.

Multi-agent systems further extend this idea by assigning different roles to multiple LLM agents, such as debate Du et al. (2023), collaboration Li et al. (2023); Zhang et al. (2024), software teams Qian et al. (2024); Hong et al. (2024), general multi-agent frameworks Wu et al. (2023); Chen et al. (2024), and agent construction tools Wu et al. (2024b). However, these systems usually depend on human-designed roles, communication patterns, and workflow structures. Recent work studies automatic

workflow or agent generation, including agent creation Chen et al. (2023a), graph optimization Zhuge et al. (2024), modular search Shang et al. (2025), evolution Yuan et al. (2025), workflow search Zhang et al. (2025b); Qiao et al. (2024a), automated agent design Hu et al. (2025), and dynamic workflow graphs Niu et al. (2025). These methods reduce manual design, but they mainly aim to find or optimize a workflow that is then reused. In contrast, our method focuses on *adaptive inference-time workflow routing*: instead of selecting one fixed workflow, it repeatedly decides what to do next based on the current intermediate answer and workflow state.

2.2 Efficient Multi-Agent Routing

Another related direction improves efficiency by allocating LLM computation selectively. In model-level routing, FrugalGPT uses cascades Chen et al. (2023b), RouteLLM chooses between models Ong et al. (2025), LLM-Blender fuses model outputs Jiang et al. (2023), and Mixture-of-Agents aggregates multiple models Wang et al. (2025); Huang et al. (2024). These works improve cost-quality trade-offs, but they mainly route among models or responses rather than among workflow actions. Efficiency is especially important in multi-agent systems because extra agents, rounds, critiques, and verification steps increase cost and latency. Prior work addresses this through dynamic agent selection Liu et al. (2023), team building Song et al. (2024); Li et al. (2025), message pruning Zhang et al. (2024), asynchronous task scheduling Yu et al. (2025), and adaptive workflow allocation Niu et al. (2025); Dang et al. (2026). These methods show that multi-agent systems should not always follow a fixed or fully expanded workflow.

3 Motivation

Automatically generated agentic workflows Zhang et al. (2025b); Hu et al. (2025) can synthesize effective multi-agent pipelines, but the resulting workflows are static: the same sequence of refinement, validation, and verification is applied to every query regardless of its difficulty. As illustrated in Figure 1, this uniform treatment causes two complementary failure modes. The workflow is either **over-provisioned** for easy queries, incurring unnecessary latency and token cost, or **under-provisioned** for hard queries that require additional retrieval, verification, or reasoning beyond the searched structure.

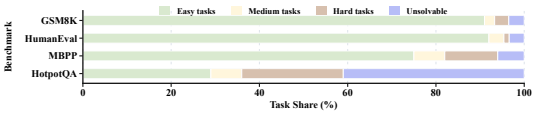


Figure 2: Portion of queries categorized as easy, medium, hard, and unsolvable on MBPP, HumanEval, HotpotQA, and GSM8K.

To quantify this mismatch, we evaluate a representative searched workflow on code-generation, reasoning, and RAG benchmarks. For each instance, we record the first-agent output, the final output of the searched workflow, and the best output among a pool of 5–8 candidate workflows augmented with additional verification, rethinking, or re-retrieval steps. Using benchmark correctness as an oracle, we classify each query as *Easy* (first-agent output already correct), *Medium* (default workflow needed), *Hard* (default fails but a candidate workflow succeeds), or *Unsolvable* (no workflow succeeds).

As shown in Figure 2, task difficulty is highly skewed. On MBPP and HumanEval, over 80% of instances are solved by the first agent alone, while fewer than 6% actually need the full fixed workflow. Meanwhile, on HotpotQA, nearly 23% of instances cannot be solved by the searched workflow but are solved by candidates with additional verification or retrieval steps.

These results lead to three key takeaways:

1. **Most queries are over-provisioned.** For the majority of easy instances, the searched workflow is overly complex, adding cost and latency without improving correctness.
2. **A non-trivial fraction of queries are under-provisioned.** For harder instances, the searched workflow lacks the verification and validation steps needed to succeed.
3. **Static workflows are the root cause.** Prior methods commit to a single fixed workflow for all queries, motivating a *dynamic* workflow design that selects an appropriate workflow based on each query's difficulty and intermediate outputs.

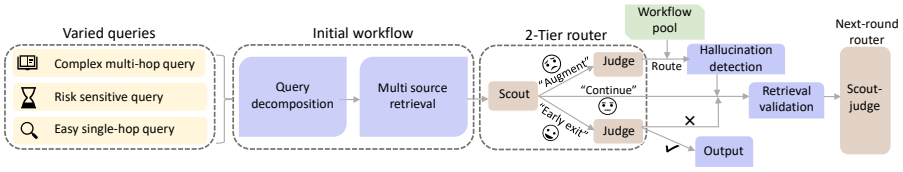


Figure 3: Overview of the DynaMAS framework. The Scout performs a low-cost three-way classification of each intermediate output. The Judge is invoked only on EARLYEXIT and AUGMENT states, where it either verifies the output or chooses a workflow from the candidate pool to augment the current workflow.

4 METHOD

We propose **DynaMAS**, a **Dyna**mic workflow routing framework for **M**ulti-**A**gent **S**ystems that operates on the intermediate output produced at each step of a multi-agent LLM pipeline. Unlike prior learned routers Zhang et al. (2025a); Yue et al. (2025); Ong et al. (2025), which require a dedicated training stage and must be retrained whenever the task distribution, agent pool, or dataset shifts, DynaMAS is *training-free*: it relies on fixed deterministic checkers, zero-shot LLM prompting, and an online-maintained decision tree and workflow library. As a result, the framework transfers across datasets and domains without any gradient updates.

The framework is organized as a two-tier cascade with explicit, asymmetric roles:

- The **Scout** (Section 4.1) is a lightweight module composed of deterministic Python checkers and a small language model $\mathcal{M}_S$. It produces a fast three-way classification of the current intermediate output: CONTINUE, EARLYEXIT, or AUGMENT.
- The **Judge** (Section 4.2) is a stronger LLM $\mathcal{M}_L$ invoked *only* on EARLYEXIT or AUGMENT. On EARLYEXIT it acts as a confirmatory verifier; on AUGMENT it selects a corrective workflow from a dynamically maintained candidate set, guided by an online-learned failure-type mapping.

Let q be the user query and $h_t = (a_1, o_1, \ldots, a_t, o_t)$ the trace of agent calls and outputs through step t. We refer to $s_t = (q, h_t, o_t)$ as the *state* at the t-th routing checkpoint, and let $\mathbf{f}_t = [\mathbf{f}_t^{\text{rule}}, \mathbf{f}_t^{\text{lm}}]$ denote the feature vector the Scout extracts from s_t. The dominant cost—a Judge invocation—is paid only when it materially affects the outcome: CONTINUE states are dispatched directly to a default refinement, while EARLYEXIT and AUGMENT are routed through the Judge.

4.1 SCOUT: TIER-1 CLASSIFICATION

The Scout combines deterministic rule-based checkers with a lightweight LM-based rubric estimator. All deterministic checkers run in parallel and complete in under 0.3s, as shown in Section 5.3.

4.1.1 RULE-BASED CHECKERS

Local Consistency ϕ_{cons}**.** Given k candidate outputs $\{o_t^{(j)}\}_{j=1}^{k}$ produced by parallel agents or temperature-τ sampling,

$$\phi_{\text{cons}}(s_t) = \frac{2}{k(k-1)} \sum_{i<j} \text{sim}(o_t^{(i)}, o_t^{(j)}),$$

where sim is exact match (short answers) or a weighted combination of normalized edit distance, ROUGE-L, and embedding cosine similarity (free-form answers). When $k = 1$, $\phi_{\text{cons}} = 1$, indicating the absence of observable disagreement rather than positive evidence of correctness.

Agent Reliability ϕ_{hist}**.** For each agent $a_i \in h_t$, Scout maintains an EMA reliability estimate based on whether prior outputs survived downstream verification:

$$\phi_{\text{hist}}^{(n+1)}(a_i) = (1-\eta)\phi_{\text{hist}}^{(n)}(a_i) + \eta \mathbb{1}[\text{verified}], \quad \eta \in (0,1).$$

The checker summarizes the trace via the minimum, mean, and final-step agent reliabilities.

Format Compliance $\phi_{\text{format}} \in \{0, 1\}$**.** Returns 1 iff the output satisfies all observable task constraints. Concrete checks are task-specific: parsable final answer and preserved units for math; required signature, language, and syntactic parseability for code; valid citation format and resolvable evidence identifiers for RAG. These checks identify low-cost specification violations that should prevent early exit, not prove semantic correctness.

4.1.2 SMALL-LM RUBRIC ESTIMATOR

The small LM $\mathcal{M}_S$ is prompted zero-shot with a fixed rubric on $(q, o_t, \mathbf{f}_t^{\text{rule}})$ and returns a bounded JSON object $\mathbf{f}_t^{\text{lm}} = (\phi_{\text{LM}}, \sigma_t)$, where $\phi_{\text{LM}} \in [0, 1]$ is the estimated reliability of the current output and σ_t is a categorical symptom label drawn from a bounded vocabulary Σ. The output is capped at 150 tokens. The Scout prompt is included in section E.

4.1.3 THREE-WAY CLASSIFICATION

Scout combines all signals into an aggregate utility

$$u_t = w_1\phi_{\text{cons}} + w_2\phi_{\text{hist}} + w_3\phi_{\text{format}} + w_4\phi_{\text{LM}},$$

and decides

$$g(\mathbf{f}_t) = \begin{cases} \textsc{EarlyExit} & u_t \geq \tau_{\text{good}} \text{ and } \phi_{\text{format}} = 1, \\ \textsc{Continue} & \tau_{\text{good}} > u_t \geq \tau_{\text{risk}} \text{ and } \phi_{\text{format}} = 1, \\ \textsc{Augment} & u_t < \tau_{\text{risk}} \text{ or } \phi_{\text{format}} = 0. \end{cases}$$

The thresholds $\tau_{\text{risk}}, \tau_{\text{good}}$ are calibrated on a held-out set via split-conformal prediction. Scout exits early only when the output has high estimated utility *and* satisfies all format constraints; it triggers augmentation when quality is low or constraints are violated. We discuss parameter selection in subsection B.1.

4.2 JUDGE: TRAINING-FREE ONLINE ROUTING

The Judge $\mathcal{M}_L$ plays asymmetric roles on the two non-CONTINUE classes:

- On EARLYEXIT, it confirms whether o_t is genuinely sufficient, returning either STOP or a conservative fallback. This double-check prevents fluent but under-grounded outputs from terminating the workflow.
- On AUGMENT, it selects a corrective operator $a_{t+1} \in \mathcal{A}_t$ from the candidate set to augment the current workflow.

Why training-free. Existing routing methods such as the multi-agent router of Zhang et al. (2025a) train a dedicated routing model on labeled traces. This design ties the router to a specific dataset and agent topology, and migrating to a new domain requires re-collecting traces and retraining. We instead support routing through two training-free online structures, both maintained by a parallel summarizer LLM $\mathcal{M}_P$ that runs asynchronously *off the critical inference path*: a *failure-type mapping* translating Scout signals into likely failure modes, and a *candidate workflow set* that grows from observed failures. Both adapt continuously as new queries arrive, without gradients, fine-tuning, or task-specific labels.

4.2.1 ONLINE-LEARNED FAILURE-TYPE MAPPING

The Scout's symptom σ_t reflects only dataset-agnostic checks. To incorporate dataset-specific patterns, we maintain a mapping M from Scout signals to a distribution over failure modes, implemented as an *ordered decision tree* (Figure 4). The tree is rooted at the categorical label σ_t (a multi-way split over Σ) and refines downward through ϕ_{LM}, ϕ_{format}, ϕ_{cons}, and ϕ_{hist} via binary threshold splits. Restricting deeper levels to binary splits keeps threshold selection one-dimensional and is sufficient given the bounded resolution of the underlying signals. Full details are in Appendix A.

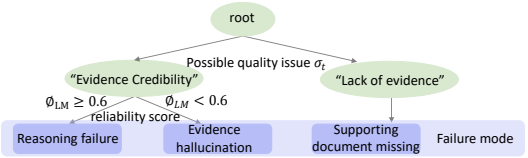


Figure 4: Example failure-type mapping tree.

Lookup. Given signals $(\sigma_t, \phi_{\text{LM}}, \phi_{\text{format}}, \phi_{\text{cons}}, \phi_{\text{hist}})$, we walk top-down: follow the σ_t-indexed child at the root, then at each internal node v step left if the corresponding feature is $\leq \theta_v$ and right otherwise, until reaching a leaf whose empirical label distribution defines $\hat{\mathcal{T}}_t$.

Update. The tree is initialized from a small set of pilot runs that yield the initial failure-mode inventory $\mathcal{T}$ and a coarse skeleton. Online, each completed query is labeled with its true failure type τ_T by $\mathcal{M}_P$, contributing one observation per Scout checkpoint along the corresponding root-to-leaf path. Thresholds are periodically re-fit by information gain and leaves with sufficient samples are promoted to internal nodes. *No gradients or retraining are involved*, so the same procedure applies unchanged to a new dataset.

4.2.2 Online-Expanded Candidate Workflow Set

The candidate set at time t is the union of a predefined augmentation library $\mathcal{A}_0$ and a growing synthesized set $\mathcal{A}_t^{\text{syn}}$:

$$\mathcal{A}_t \;=\; \mathcal{A}_0 \;\cup\; \mathcal{A}_t^{\text{syn}}.$$

$\mathcal{A}_0$ is seeded by pilot runs on the development set, and $\mathcal{A}_t^{\text{syn}}$ grows online: whenever $\mathcal{M}_P$ identifies a failure type that no existing workflow addresses, it synthesizes a new workflow in code form (following the format of Zhang et al. (2025b); Hu et al. (2025)); the candidate is added to $\mathcal{A}_t^{\text{syn}}$ if it parses, is non-duplicate, and passes a dry run (Figure 5).

In contrast to learned routers, which fix the action space at training time, this set expands automatically as new failure modes appear, allowing the framework to specialize to a new dataset with no manual intervention.

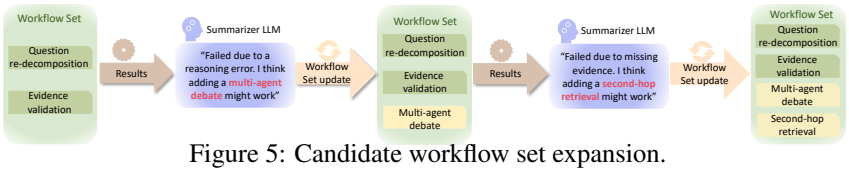


Figure 5: Candidate workflow set expansion.

4.2.3 Judge Decision

On a non-CONTINUE state, the Judge receives

$$\big(q,\ \widetilde{h}_t,\ o_t,\ \mathbf{f}_t,\ c \in \{\textsc{EarlyExit}, \textsc{Augment}\},\ \hat{\mathcal{T}}_t,\ \mathcal{A}_t\big),$$

where $\widetilde{h}_t$ is a compressed trace, $\mathbf{f}_t$ is the Scout feature vector, c is the Scout class, $\hat{\mathcal{T}}_t$ is the inferred failure-mode prior, and $\mathcal{A}_t$ is the current candidate set. Under a fixed JSON schema, the Judge (i) names the most likely failure mode (drawn from $\hat{\mathcal{T}}_t$ when $c =$ AUGMENT, or *none* when $c =$ EARLYEXIT with clean signals), (ii) selects an action $a_{t+1} \in \mathcal{A}_t$ targeting that mode, and (iii) emits a brief rationale. For EARLYEXIT, the typical decision is STOP unless residual concerns warrant a conservative fallback.

The asynchronous online updates to M and $\mathcal{A}_t^{\text{syn}}$ (Algorithm 2) run on a separate worker, off the critical inference path, and therefore do not affect end-to-end latency. Together, the decision tree and the synthesized workflow set provide a fully training-free adaptation mechanism that generalizes across datasets simply by accumulating traces.

The prompt templates for the Judge LLM and Summarizer LLM are included in section E.

5 Evaluation

5.1 Experimental Setup

Benchmarks. We evaluate SCOUT–JUDGE on four public datasets spanning three task families: HUMANEVAL Chen et al. (2021) and MBPP Austin et al. (2021) for code generation, GSM8K Cobbe et al. (2021) for mathematical reasoning, and the full-wiki setting of HOTPOTQA Yang et al. (2018)

Algorithm 1: Workflow Routing at One Checkpoint

Require: state $s_t = (q, h_t, o_t)$; checkers Φ; small LM $\mathcal{M}_S$; judge $\mathcal{M}_L$; mapping M; candidate set $\mathcal{A}_t$; thresholds $\boldsymbol{\tau}$
Ensure: next action a_{t+1}

$\mathbf{f}_t^{\text{det}} \leftarrow [\phi(s_t)]_{\phi \in \Phi}$ ▷ parallel checkers
$\mathbf{f}_t^{\text{lm}} \leftarrow \mathcal{M}_S(q, o_t, \mathbf{f}_t^{\text{det}})$ ▷ rubric + symptom σ_t
$\mathbf{f}_t \leftarrow [\mathbf{f}_t^{\text{det}}, \mathbf{f}_t^{\text{lm}}]$
$c \leftarrow g(\mathbf{f}_t; \boldsymbol{\tau})$
if $c =$ CONTINUE **then**
  return REFINE ▷ Judge bypassed
end if
$\hat{\mathcal{T}}_t \leftarrow M(\mathbf{f}_t)$
$a_{t+1} \leftarrow \mathcal{M}_L(q, \widetilde{h}_t, o_t, \mathbf{f}_t, c, \hat{\mathcal{T}}_t, \mathcal{A}_t)$
if $c =$ EARLYEXIT **and** $a_{t+1} =$ STOP **then**
  return STOP
end if
return a_{t+1}

Algorithm 2: Asynchronous Online Update

Require: trace h_T; verified outcome Y; summarizer $\mathcal{M}_P$; current mapping M; set $\mathcal{A}_{T-1}$

$\tau_T \leftarrow \mathcal{M}_P^{\text{summary}}(h_T, Y)$ ▷ label failure mode
for $t = 1, \ldots, T$ **do**
  update $M(\mathbf{f}_t)$ via EMA toward τ_T
end for
if Y unsatisfactory **and** τ_T not addressed by any $a \in \mathcal{A}_{T-1}$ **then**
  $\alpha \leftarrow \mathcal{M}_P^{\text{synthesize}}(h_T, \tau_T)$
  if α parses, is non-duplicate, passes dry run **then**
    $\mathcal{A}_T^{\text{syn}} \leftarrow \mathcal{A}_{T-1}^{\text{syn}} \cup \{\alpha\}$
  end if
end if
decay reliability statistics for all $a \in \mathcal{A}_T$

for RAG tasks. We use the official test split for final evaluation only; validation data is reserved for model selection, threshold calibration, and pilot runs that seed the failure-mode inventory $\mathcal{T}$ and initial workflow library $\mathcal{A}_0$. When a dataset has no validation split, we partition the training set 80/20.

Baselines. We compare against nine representative methods covering both hand-crafted prompting strategies and automatically designed workflows: **IO** (single-call direct prompting), **CoT** Wei et al. (2022), **Self-Refine** Madaan et al. (2023), **Reflexion** Shinn et al. (2023), **ADAS** Hu et al. (2025), **MaAS** Zhang et al. (2025a), **MasRouter** Yue et al. (2025), **Flow** Niu et al. (2025) and **AFlow** Zhang et al. (2025b). AFlow is our primary point of comparison: it is the strongest static workflow in our study and represents the setting most directly relevant to our contribution. To isolate the effect of adaptive routing, we instantiate DYNAMAS on top of the AFlow-discovered pipeline, replacing its always-execute behavior with our two-tier cascade. All methods share the same executor models and underlying agent pool; only the routing logic differs.

Implementation. Our DYNAMAS design uses three model roles. The **Judge** and **parallel summarizer** are accessed through the Volcano Engine APIs Doubao, and the **Scout small LM** is accessed through the Groq API Groq:

- **Scout small LM** $\mathcal{M}_S$: Llama 3 8B, used to produce the fast and low-overhead rubric score ϕ_{LM} and symptom label σ_t.
- **Judge** $\mathcal{M}_L$: Doubao-seed-1.6-lite, invoked only on EARLYEXIT and AUGMENT states.
- **Parallel summarizer** $\mathcal{M}_P$: Doubao-seed-2.0-pro, run asynchronously off the critical path to label failure modes and synthesize new candidate workflows. It is invoked only until the candidate workflow set saturates.

Rule-based Scout checkers are implemented in Python and run in parallel. Evaluation metrics and Scout weight calibration details are included in subsection B.1.

Table 1: Results on MBPP, HumanEval, GSM8K, and HotpotQA. *Acc./F1* denote task performance, while *Tokens* and *Lat. (s)* measure output tokens and end-to-end latency.

	MBPP			HumanEval			GSM8K			HotpotQA		
Method	Acc.	Tokens	Lat.	Acc.	Tokens	Lat.	Acc.	Tokens	Lat.	F1	Tokens	Lat.
IO	79.3	1,005	19.5	92.6	1,425	23.5	91.7	1,343	25.5	28.8	1,325	74.5
CoT	84.2	4,059	67.0	93.7	2,679	87.9	94.1	6,852	94.2	32.2	3,542	115.5
Self-Refine	82.1	3,804	79.6	92.2	6,352	93.9	92.1	8,825	115.1	35.5	9,825	290.3
Reflexion	85.5	4,362	89.7	93.1	7,264	107.2	92.9	11,250	129.5	36.1	11,069	352.5
ADAS	89.2	5,824	121.3	95.3	8,982	169.6	92.3	15,724	167.3	33.2	15,240	690.3
MaAS	91.2	4,674	105.3	93.9	6,684	120.6	94.0	12,563	157.3	39.3	11,025	393.4
MasRouter	92.3	4,832	115.2	94.0	5,967	135.2	92.1	7,625	143.3	33.7	9,582	287.9
Flow	89.5	3,274	89.3	93.1	3,395	102.6	93.5	9,782	139.3	35.3	10,887	357.9
AFlow (baseline)	<u>94.2</u>	6,643	134.3	<u>95.9</u>	9,391	171.6	<u>95.9</u>	17,583	198.3	<u>38.7</u>	12,849	515.3
Ours	**94.7**	**2,430**	**78.0**	**96.0**	**5,877**	**108.3**	**96.3**	**8,624**	**125.2**	**43.6**	**8,648**	**125.4**
Δ vs. baseline	**+0.5%**	**-63.4%**	**-41.9%**	**+0.1%**	**-37.4%**	**-36.8%**	**+0.4%**	**-50.9%**	**-36.9%**	**+4.9**	**-32.7%**	**-75.7%**

5.2 Results and analysis

Table 1 summarizes the overall performance of DynaMAS across four benchmarks. Compared with the strongest static baseline, AFlow, DynaMAS consistently reduces inference cost while preserving competitive task performance. On MBPP, HumanEval, and GSM8K, DynaMAS lowers output tokens by 63.4%, 37.4%, and 50.9%, respectively, while reducing latency by 41.9%, 36.8%, and 36.9%, with accuracy gains of 0.1–0.5%. These results indicate that dynamic early exiting can recover a large fraction of unnecessary computation without substantially sacrificing final quality.

Table 2: Per-component overhead on MBPP. Token usage and latency of the workflow agent and DynaMAS scheduling stages. Invoc. rate refers to how often a component is called at each checkpoint.

Component	Tokens	Lat.	Invoc. rate (%)
Workflow agent	1,238.5	19.82	100
Scout Unit	76.4	0.31	45.7
Judge LLM	1,125.7	17.28	15.4
Summarizer LLM	2,157.1	21.58	2.3

5.3 Overhead analysis

Table 2 shows that the additional components introduced by DynaMAS incur only modest overhead. The Scout unit is highly lightweight, requiring only 76.4 output tokens and 0.31 seconds on average, making it inexpensive to apply frequently. Although the Judge LLM is more costly, its overhead is comparable to that of a standard workflow agent and it is invoked only when the Scout determines that deeper reasoning is needed. Thus, many easy examples can exit early without paying this additional cost. The Summarizer LLM has higher token usage and latency, but it is called after a task is finished and is no longer invoked once the candidate workflow set is saturated, keeping its overall overhead low. Overall, the conditional execution design keeps average token consumption and latency low while improving efficiency without substantial additional cost.

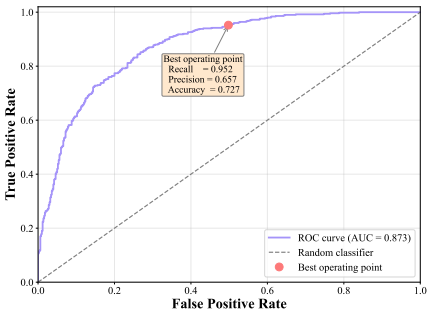


Figure 6: ROC curve of the Scout unit.

5.4 Prediction accuracy analysis

Tier-1 Scout prediction accuracy To evaluate Tier-1 Scout prediction accuracy, we assign each intermediate artifact an aggregate utility score and sweep a threshold τ: outputs scoring at least τ are accepted via early exit, while the rest continue to the next stage. On MBPP, we treat early-exited outputs as positive predictions, compare them against unit-test correctness, and sweep τ to obtain the ROC curve in Figure 6. The best operating point reaches 72.7% accuracy, with high recall (95.2%) but lower precision (65.7%), indicating that Scout is effective as a conservative trigger while the Judge handles borderline cases to reduce unsafe early exits.

Table 3: Tier-2 routing accuracy. Top-1 matches the oracle (lowest-latency successful) workflow; Top-2 also counts a match with the second-fastest. Latency increase is averaged over all AUGMENT checkpoints, including correct picks.

Benchmark	**Top-1**	**Top-2**	**Lat. Inc.**
GSM8K	87.2	95.2	11.2%
MBPP	77.5	83.8	16.5%
HotpotQA	74.7	78.5	18.6%

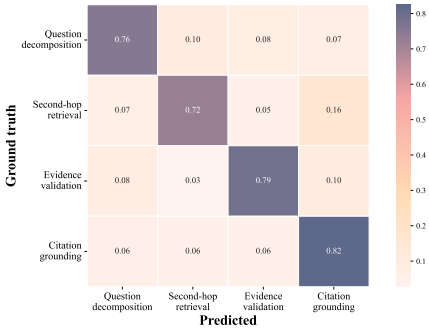


Figure 7: Confusion matrix for Judge routing on a four-workflow candidate set.

Tier-2 Judge routing accuracy. We evaluate the Judge's routing quality on every checkpoint the Scout marks as AUGMENT. For each such instance, we execute all candidate workflows and define the *oracle* as the lowest-latency workflow that successfully solves it, where success follows the benchmark metric (exact match for math, unit-test pass rate for code, evidence-supported correctness for RAG). The Judge's choice is scored Top-1 if it matches the oracle and Top-2 if it matches the oracle or the next-fastest successful workflow. As shown in Table 3 and Figure 7, the Judge attains an average Top-1 of 79.8% and Top-2 of 85.8% across the three benchmarks. The gap between Top-1 and Top-2 indicates that most routing errors fall between workflows of

similar expected utility rather than between clearly correct and clearly incorrect recovery strategies. Accuracy is highest on GSM8K, where Scout symptoms map cleanly onto a small set of recovery actions (re-derivation, self-consistency voting), and lowest on HotpotQA, where retrieval gaps, citation repair, and broader rewriting compete for the same signals. Code generation lies in between: traceback-driven repair, adversarial test synthesis, and fuzzing are all plausible responses to a failing program, making the routing target inherently ambiguous. Even when the Judge misses the oracle, its choice typically offers a comparable utility–latency trade-off—the average normalized regret is 0.045.

5.5 Ablation Study

We isolate the contribution of each tier on HotpotQA. The **Scout-only** variant uses the four Scout signals to decide between early-exit and continuation, bypassing the Judge entirely. It achieves the lowest latency and token cost, but its lightweight checks are insufficient on harder instances, leading to a 1.48 drop in F1 relative to the baseline. The **Judge-only** variant invokes $\mathcal{M}_L$ at every checkpoint without Scout filtering. This yields the highest F1 (43.94) but at near-baseline cost, since every intermediate output triggers an expensive Judge call.

Table 4: Two-tier module ablation on HotpotQA. *Scout only* disables the Judge; *Judge only* invokes $\mathcal{M}_L$ on every checkpoint without Scout filtering; the full cascade combines both.

Method	F1	Tokens	Lat.
Baseline	38.72	12,849	515.25
Scout only	37.24	9,548	**94.25**
Judge only	**43.94**	12,532	437.45
Scout + Judge	43.63	**8,648**	125.36

The full **Scout+Judge** cascade captures the strengths of both tiers: Scout filters confident cases and routes only uncertain ones to the Judge. The result is F1 within 0.31 of the Judge-only ceiling while reducing latency by 75.7% and tokens by 32.7% relative to the baseline. This confirms our central design hypothesis — that paying the cost of a strong Judge is only worthwhile on the small subset of states where it materially affects the outcome.

5.6 Analysis of the Online Learning Process

We analyze how the online learning process evolves with more processed queries by tracking two quantities: the synthesized candidate set size $|\mathcal{A}_t^{\text{syn}}|$, and the routing quality $Q_t = L(w^*)/L(\hat{w}_t) \in (0, 1]$, where $\hat{w}_t$ is the Judge's choice at query t, w^* is the lowest-latency successful workflow in the *final* candidate set, and $L(\cdot)$ denotes latency. Higher Q_t indicates more efficient routing.

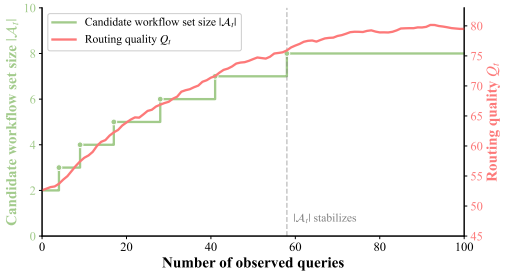


Figure 8: Workflow candidate set size and routing quality change with the number of queries on HotpotQA.

As shown in Figure 8, both curves exhibit a fast-rise-then-saturate pattern. The candidate set grows rapidly during the first ∼60 queries and then plateaus, indicating that most recurring failure modes are covered after only a small number of observed failures. Routing quality follows a similar trend, saturating around ∼80 queries. These results show that the online mechanism adapts efficiently, requiring only tens of examples to construct a useful workflow space and an effective routing prior, supporting the training-free design. A more detailed explanation is provided in Appendix B.2.

6 Conclusion

We introduced DynaMAS, a training-free framework for dynamic, instance-level workflow routing in multi-agent systems, pairing a lightweight *Scout* that runs at every step with a stronger LLM-based *Judge* invoked only on borderline cases, plus a parallel summarizer that grows the candidate workflow set online. Across four benchmarks, DynaMAS cuts tokens by 46.1% and latency by 47.8% with no accuracy loss relative to the strongest static baseline. Future work will extend the framework to open-ended generation tasks where verifiable outcomes are unavailable.

AI USE STATEMENT

Generative AI models were used as experimental components of DynaMAS, as described in the main paper, to instantiate, assess, and adapt candidate workflows. Generative AI tools were also used to assist with language editing, proofreading, and LaTeX formatting of the manuscript. No generative AI system was used to fabricate or alter experimental data, numerical results, citations, or scientific claims. All authors reviewed and verified the final manuscript, methods, code, analyses, and reported results, and take full responsibility for the content of this submission.

ETHICS STATEMENT

This work studies resource-efficient dynamic workflow routing in controlled benchmark settings, including code generation and retrieval-augmented generation tasks. It does not involve human participants or the collection of private personal data. The experiments use established benchmark datasets and third-party model APIs in accordance with their applicable terms of use.

DynaMAS can synthesize and select workflows online; if such systems are deployed beyond controlled evaluation, erroneous outputs, unsafe tool use, privacy exposure, and feedback-induced bias may create risks. Therefore, practical deployments should use appropriate access controls, isolated execution environments, monitoring and audit logs, data-minimization practices, and human approval for consequential actions. We do not claim that DynaMAS is suitable for high-stakes or safety-critical deployment without these safeguards.

REPRODUCIBILITY STATEMENT

The paper and appendix specify the DynaMAS formulation, workflow-routing and verification procedures, benchmark protocols, token-accounting methodology, model roles, and evaluation metrics. Upon acceptance, we will publicly release all source code and all data required to reproduce the study, including workflow implementations, prompts, configurations, seeds, preprocessing scripts, evaluation scripts, workflow-library snapshots, and experiment logs/results. For any third-party benchmark data that cannot legally be redistributed, we will provide the exact version identifiers, licensing information, and automated download and preprocessing scripts needed to reconstruct the released experimental inputs. We will also document the model/API configurations and runtime settings used in the experiments to facilitate faithful reproduction and extension of our results.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023a.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, et al. Multi-agent collaboration via evolving orchestration. *Advances in neural information processing systems*, 38:165025–165059, 2026.

Doubao. Doubao-seed-1-6. `https://www.volcengine.com/product/doubao`.

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.

Peiyuan Feng, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. Agile: A novel reinforcement learning framework of llm agents. *Advances in Neural Information Processing Systems*, 37:5244–5284, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Groq. Groq api. `https://groq.com/`.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 8154–8173, 2023.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Ting Liu, and Bing Qin. Ensemble learning for heterogeneous large language models with deep parallel collaboration. *Advances in Neural Information Processing Systems*, 37:119838–119860, 2024.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, 2023.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pp. 3301–3310. PMLR, 2019.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*, 2023.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 19495–19517, 2025.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Siddharth Nayak, Adelmo M Orozco, Marina T Have, Vittal Thirumalai, Jackson Zhang, Darren Chen, Aditya Kapoor, Eric Robinson, Karthik Gopalakrishnan, James Harrison, et al. Long-horizon planning for multi-agent robots in partially observable environments. *Advances in Neural Information Processing Systems*, 37:67929–67967, 2024.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. In *International Conference on Learning Representations*, volume 2025, pp. 74949–74977, 2025.

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

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. *arXiv preprint arXiv:2410.07869*, 2024a.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. *Advances in Neural Information Processing Systems*, 37:114843–114871, 2024b.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. Adaptive in-conversation team building for language model agents. *arXiv preprint arXiv:2405.19425*, 2024.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pp. 33944–33963, 2025.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, et al. Can graph learning improve planning in llm-based agents? *Advances in Neural Information Processing Systems*, 37:5338–5383, 2024a.

Yue Wu, Yewen Fan, So Yeon Min, Shrimai Prabhumoye, Stephen McAleer, Yonatan Bisk, Ruslan Salakhutdinov, Yuanzhi Li, and Tom Mitchell. Agentkit: Structured llm reasoning with dynamic graphs. *arXiv preprint arXiv:2404.11483*, 2024b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Junwei Yu, Yepeng Ding, and Hiroyuki Sato. Dyntaskmas: A dynamic task graph-driven framework for asynchronous and parallel llm-based multi-agent systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 35, pp. 288–296, 2025.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, 2025.

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.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

Xiaoling Zhou, Ou Wu, and Nan Yang. Class and attribute-aware logit adjustment for generalized long-tail learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 22991–22999, 2025.

Yifei Zhou and Andrea Zanette. Archer: training language model agents via hierarchical multi-turn rl. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 62178–62209, 2024.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A ONLINE-LEARNED FAILURE-TYPE MAPPING

We implement the mapping M as an incrementally grown *ordered decision tree* over the Scout signals, where the splitting order is fixed in advance by the discriminative power of each feature. The tree is rooted at the symptom label σ_t produced by the small-LM rubric (the most semantically informative signal) and refines downward through ϕ_{LM}, ϕ_{format}, ϕ_{cons}, and ϕ_{hist} in that order. The root is a multi-way split on the categorical label σ_t, with one child per element of the bounded symptom vocabulary Σ. All deeper levels are *binary* splits, each parameterized by a single threshold θ_v on the corresponding scalar feature: a state descends left if its feature value is $\leq \theta_v$ and right otherwise. Restricting deeper levels to two-way splits keeps the branching factor small, reduces threshold selection to a one-dimensional search over a finite set of candidates, and is sufficient in practice given the bounded resolution of the underlying signals.

Tree structure. Let $\mathcal{V}$ denote the set of nodes currently materialized. Each node $v \in \mathcal{V}$ stores

$$v \;=\; \big(\, \mathbf{n}_v \in \mathbb{R}^{|\mathcal{T}|}_{\geq 0},\;\; N_v = \sum_{\tau} n_{v,\tau},\;\; \theta_v,\;\; \mathrm{children}(v)\,\big),$$

where $\mathbf{n}_v$ is an exponentially decayed pseudo-count vector over the failure inventory $\mathcal{T}$ for the sub-population of states reaching v, θ_v is the threshold used to route observations at v (set when v is split, undefined at leaves, and replaced by the categorical σ_t at the root), and $\mathrm{children}(v)$ is the dictionary of materialized children — one entry per $\sigma \in \Sigma$ at the root, and at most a left/right pair at all deeper levels. The root holds the global marginal; the depth of the tree is at most five, equal to the number of features.

Path lookup. Given a Scout signal $(\sigma_t, \phi_{\mathrm{LM}}, \phi_{\mathrm{format}}, \phi_{\mathrm{cons}}, \phi_{\mathrm{hist}})$, the path $\pi_t = (v_0, v_1, \ldots, v_{d_t})$ is determined by descending from the root: at v_0 we select the child indexed by σ_t, and at each subsequent v_j we descend left if the corresponding feature value is $\leq \theta_{v_j}$ and right otherwise. Descent halts at the deepest node whose required child has been materialized. The estimated failure-type distribution is then obtained by interpolating from coarse to fine along π_t:

$$\hat{p}_t(\tau) \;=\; \sum_{j=0}^{d_t} \omega_j \, \frac{\alpha_\tau + n_{v_j,\tau}}{\alpha_0 + N_{v_j}}, \qquad \omega_j = \lambda_j \prod_{i=j+1}^{d_t} (1-\lambda_i), \quad \lambda_j = \frac{N_{v_j}}{N_{v_j} + \kappa},$$

where $\boldsymbol{\alpha}$ is a Dirichlet prior on $\mathcal{T}$ and κ controls how aggressively a sparse node defers to its parent. A well-populated descendant trusts itself; a thinly-populated one borrows strength from its ancestors all the way up to the global marginal at the root. The top-K types under $\hat{p}_t$ are returned as $\hat{\mathcal{T}}_t$.

Online update. After each completed query with verified failure label τ_T produced by $\mathcal{M}_P$, every Scout checkpoint t on the trace contributes one update along its path. For each node $v_j \in \pi_t$,

$$\mathbf{n}_{v_j} \;\leftarrow\; (1-\eta_M)\,\mathbf{n}_{v_j} \;+\; \eta_M\,\mathbf{e}_{\tau_T},$$

where $\mathbf{e}_{\tau_T}$ is the one-hot indicator and $\eta_M \in (0,1]$ is the learning rate. Updates touch only $d_t + 1 \leq 6$ nodes per checkpoint and require no gradients.

Online growth. A non-root leaf v at depth $j < 5$ is split once two conditions hold simultaneously: (i) sufficient evidence, $N_v \geq n_{\mathrm{split}}$, and (ii) the existence of a threshold on the level-$(j{+}1)$ feature that produces informative children. To this end, v maintains a small ring buffer $\mathcal{B}_v$ of recent observations $\{(\phi_i^{(j+1)}, \tau_i)\}$. The candidate threshold set $\Theta^{(j+1)}$ consists of the bin boundaries of the corresponding feature ($\Theta = \{0.1, 0.2, \ldots, 0.9\}$ for each of $\phi_{\mathrm{LM}}, \phi_{\mathrm{cons}}, \phi_{\mathrm{hist}}$ at 0.1 resolution, and the singleton $\{0.5\}$ for the already-binary ϕ_{format}). We choose

$$\theta_v^* \;=\; \arg\max_{\theta \in \Theta^{(j+1)}} \; \tfrac{N_L(\theta)}{N}\, D_{\mathrm{KL}}\big(p_{L,\theta} \,\|\, p_v\big) + \tfrac{N_R(\theta)}{N}\, D_{\mathrm{KL}}\big(p_{R,\theta} \,\|\, p_v\big),$$

where $p_{L,\theta}, p_{R,\theta}$ are the empirical failure-type distributions on either side of θ in $\mathcal{B}_v$ and N_L, N_R are the corresponding counts. The split is admitted only if this maximum exceeds an informativeness threshold δ_{split}; otherwise v remains a leaf and the buffer continues to grow. Because $|\Theta^{(j+1)}| \leq 9$, the search is exhaustive and trivial to perform online. Symmetrically, a child whose distribution drifts back toward its parent's ($D_{\mathrm{KL}} < \delta_{\mathrm{prune}}$) over a sliding window is pruned and its counts are folded back into the parent. Both thresholds are robust: in our deployment we set $n_{\mathrm{split}} = 50$, $\delta_{\mathrm{split}} = 0.1$, and $\delta_{\mathrm{prune}} = 0.02$.

Table 5: Gate configuration across datasets. Learned component weights and decision thresholds used by the gate.

Dataset	$\mathbf{w}_1(\phi_{cons})$	$\mathbf{w}_2(\phi_{hist})$	$\mathbf{w}_3(\phi_{format})$	$\mathbf{w}_4(\phi_{LM})$	$\boldsymbol{\tau}_{\text{good}}$	$\boldsymbol{\tau}_{\text{risk}}$
MBPP	0.07	0.02	0.06	0.85	0.80	0.23
HumanEval	0.07	0.04	0.07	0.82	0.82	0.15
GSM8K	0.14	0.11	0.00	0.75	0.85	0.37
HotpotQA	0.02	0.03	0.04	0.91	0.75	0.34

Properties. The tree-based mapping has three properties that make it well-suited to this setting. First, it is *strictly online*: lookups and updates are $O(d) \leq O(5)$, no retraining is required, and growth happens incrementally as deployment data accumulates. Second, it is *naturally hierarchical*: cold-start states with novel signal combinations fall back to coarser ancestor statistics rather than to a uniform prior, so the Judge receives a meaningful $\hat{\mathcal{T}}_t$ even on the first encounter with a particular signal pattern. Third, the depth-first feature ordering — symptom string, then LM rubric confidence, then format compliance, then consistency, then agent history — exposes the most semantically informative axis at the top, so even shallow trees (depth two or three) already provide useful priors before the tail of the distribution is well sampled. The restriction to binary splits below the root keeps the branching factor small, makes threshold selection a one-dimensional search over at most nine candidates, and ensures that subtrees never fragment data more aggressively than the underlying signal resolution warrants.

B Evaluation details

B.1 Experiment settings

Metrics. We report *Solve Rate* (%) on GSM8K and *pass@1* on HumanEval and MBPP; for HotpotQA we report the final answer F1 on the predicted answer. To capture inference cost, we additionally report output token consumption (*Tokens*) and end-to-end wall-clock latency in seconds (*Lat.*), since output tokens typically dominate cost in multi-step agentic workflows. All numbers are averaged over five independent runs.

Calibration. The Scout aggregation weights (w_1, w_2, w_3, w_4) for $(\phi_{\text{cons}}, \phi_{\text{hist}}, \phi_{\text{format}}, \phi_{\text{LM}})$ and the two decision thresholds $(\tau_{\text{risk}}, \tau_{\text{good}})$ are selected by grid search on the validation set; final values are reported in Table 5. Because GSM8K answers are single numerical values, we drop the format-compliance term ($w_3 = 0$) and instead require each agent to emit a short rationale alongside the answer, which the Scout small LM uses when computing ϕ_{LM}. Thresholds are recalibrated per dataset using split-conformal prediction; no weights or model parameters are otherwise modified across datasets, consistent with the training-free design of the framework.

B.2 Analysis of the Online Learning Process

We analyze how the online learning process evolves with more processed queries by tracking two quantities: the size of the candidate workflow set $\mathcal{A}_t^{\text{syn}}$ and the Tier-2 Judge's routing quality. We define it at query t as

$$Q_t = \frac{L(w^*)}{L(\hat{w}_t)}, \tag{1}$$

where $\hat{w}_t$ is the workflow selected by the Judge at the current stage of online learning, w^* is the lowest-latency successful workflow in the final candidate set, and $L(\cdot)$ denotes workflow latency. A workflow is considered successful if it solves the instance under the benchmark metric. Under this definition, $Q_t \in (0, 1]$, with values closer to 1 indicating more efficient routing.

As shown in Figure 8, both curves exhibit a fast-rise-then-saturate pattern. The candidate set grows rapidly during the first $\sim$60 queries and then plateaus, indicating that most recurring failure modes are covered after only a small number of observed failures. Routing quality follows a similar trend, saturating around $\sim$80 queries: as $\mathcal{A}_t^{\text{syn}}$ becomes more expressive and the failure-type mapping accumulates more evidence, the Judge gains both better recovery options and a sharper prior over failure modes. Beyond this point, additional queries yield diminishing returns.

Table 6: Five-run mean ± standard deviation for task performance, output tokens, and latency.

	Performance		Tokens		Latency (s)	
Benchmark	**AFlow**	**DynaMAS**	**AFlow**	**DynaMAS**	**AFlow**	**DynaMAS**
MBPP	94.2±0.31	94.7±0.28	6643±179	2430±76	134.3±4.3	78.0±3.1
HumanEval	95.9±0.30	96.0±0.27	9391±246	5877±166	171.6±5.6	108.3±4.4
GSM8K	95.9±0.35	96.3±0.33	17583±451	8624±245	198.3±6.2	125.2±5.0
HotpotQA	38.7±0.62	43.6±0.57	12849±342	8648±230	515.3±16.4	125.4±5.2

Table 7: Input (I), output (O), and total (T) token accounting.

	MBPP			HumanEval			GSM8K			HotpotQA		
Method	I	O	T	I	O	T	I	O	T	I	O	T
IO	234	1005	1239	425	1425	1850	143	1343	1486	624	1325	1949
CoT	281	4059	4340	651	2679	3330	199	6852	7051	892	3542	4434
Self-Refine	1793	3804	5597	2988	6352	9340	1292	8825	10117	4115	9825	13940
Reflexion	2535	4362	6897	4252	7264	11516	2933	11250	14183	7270	11069	18339
ADAS	8891	5824	14715	11241	8982	20223	10103	15724	25827	17739	15240	32979
MaAS	2739	4674	7413	4612	6684	11296	5675	12563	18238	5102	11025	16127
MasRouter	1985	4832	6817	2481	5967	8448	3286	7625	10911	4770	9582	14352
Flow	2958	3274	6232	3125	3395	6520	3345	9782	13127	8082	10887	18969
AFlow	3876	6643	10519	5892	9391	15283	4305	17583	21888	5424	12849	18273
DynaMAS	1772	2430	4202	3914	5877	9791	2534	8624	11158	4125	8648	12773
DynaMAS vs. AFlow	−60.1%			−35.9%			−49.0%			−30.1%		

These results show that the online mechanism adapts efficiently, requiring only tens of examples to construct a useful workflow space and an effective routing prior — an important property given the training-free design of the framework. This analysis complements the main discussion in Section 5.6.

C Additional Evaluations

This section reports the additional evaluations requested in the rebuttal. Unless otherwise noted, values are means over five independent runs. Calibration, pilot seeding, candidate-workflow admission, and threshold selection use validation data only and are frozen before held-out test evaluation.

C.1 Run-to-run variability

Table 6 quantifies the stability of the main comparison over five independent end-to-end runs. It reports variation in task performance as well as in output tokens and latency, so that the reported efficiency gains can be interpreted alongside their run-to-run dispersion.

C.2 Complete token accounting

The main results report output tokens, which capture the dominant cost of the multi-step workflows. Table 7 additionally separates input (I), output (O), and total ($T = I + O$) tokens to make the full cost comparison transparent.

C.3 Routing decisions and verification

This analysis characterizes how the cascade allocates computation across held-out checkpoints. Table 8 reports the Scout's three-way decisions and the Judge's subsequent verification of proposed early exits; Table 9 isolates the value of that verification step.

Across the four benchmarks, the Scout assigns 43.5% of checkpoints to Continue, 32.2% to EarlyExit, and 24.3% to Augment on average. The Judge confirms 91.6% of early-exit proposals and overrides the remaining 8.4%.

Table 8: Scout decisions and Judge verification on held-out test sets.

Benchmark	Scout decision (%)			Judge (%)	
	Continue	EarlyExit	Augment	Confirm	Override
MBPP	40.4	35.5	24.1	90.1	9.9
HumanEval	45.2	31.6	23.2	93.5	6.5
GSM8K	51.3	21.5	27.2	88.6	11.4
HotpotQA	37.2	40.1	22.7	94.3	5.7
Average	43.5	32.2	24.3	91.6	8.4

Table 9: Ablation of Judge verification on the non-AUGMENT path.

	Performance		Output tokens		Latency (s)	
Benchmark	**DynaMAS**	**NoVerify**	**DynaMAS**	**NoVerify**	**DynaMAS**	**NoVerify**
MBPP	94.7	91.4	2430	2280	78.0	75.9
HumanEval	96.0	94.0	5877	5465	108.3	104.7
GSM8K	96.3	94.3	8624	8225	125.2	119.1
HotpotQA	43.6	39.2	8648	8192	125.4	117.7
Average change	-2.9 points		-5.8%		-4.3%	

Table 10: Controlled comparison with a shared candidate workflow set $\mathcal{A}_0$.

Benchmark	Router	Performance	Total tokens	Latency (s)
MBPP	MasRouter	89.8 ± 0.56	8910 ± 292	123.6 ± 5.0
	DynaMAS A0	91.3 ± 0.41	5500 ± 172	99.4 ± 3.7
HotpotQA	MasRouter	32.2 ± 0.72	18240 ± 552	298.8 ± 7.6
	DynaMAS A0	38.5 ± 0.64	16340 ± 452	154.5 ± 5.8

Table 11: Online synthesis quality and long-horizon stability.

Benchmark	Proposed	Nondup.	Admitted	Picked	Rate
MBPP	7.2	6.8	6.4	5.8	90.6%
HotpotQA	8.4	7.8	7.6	7.0	92.1%
Average	7.8	7.3	7.0	6.4	91.4%

C.4 SEPARATING ROUTING FROM WORKFLOW EXPANSION

With a common candidate set $\mathcal{A}_0$, DynaMAS A0 improves over MasRouter by 1.5 points on MBPP and 6.3 F1 points on HotpotQA, while reducing total tokens by 38.3% and 10.4%, respectively.

C.5 ONLINE SYNTHESIS, CALIBRATION, AND SCALABILITY

The synthesis funnel admits 89.7% of proposed workflows on average, and 91.4% of admitted workflows are selected at least once. The first two synthesized workflows improve MBPP accuracy from 86.3% to 92.1% and HotpotQA F1 from 36.1 to 42.0. On a 250-query warm-up trace, the nonduplicate candidate set saturates at 8.6 and accuracy remains at 94.8%.

Experiments use two Intel Xeon 6520P CPUs with 24 physical cores each and 256 GB RAM. Eight CPU workers handle rule-based checks and workflow scheduling; the Scout is served through Groq and the Judge and summarizer through Volcano Engine, with no local GPU. Scout latency remains between 0.29 and 0.31 seconds as the candidate pool grows from 0 to 10 workflows. After approximately 80 queries, the nonduplicate candidate set saturates at 8.6 and accuracy remains 94.8% through 250 queries.

Interpretation and limitations. Training-free does not mean calibration-free. Thresholds, the failure vocabulary, and the initial candidate library are calibrated on a held-out validation split and frozen before testing; substantial distribution shift may make them stale. The present experiments

evaluate the complete Scout feature bundle rather than a complete one-signal-at-a-time checker ablation, so individual checker effects remain a direction for future work. Lower inference cost does not remove the need for safety, access-control, monitoring, and human-review policies.

D LIMITATIONS

Cold-start period. The online mapping M and synthesized workflow set $\mathcal{A}_t^{\text{syn}}$ require accumulated traces to stabilize: Figure 8 shows saturation only after $\sim$60–80 queries. Pilot seeding mitigates this but does not eliminate it; deployments that see only a handful of queries from a new distribution operate in a transient regime where routing decisions can be noticeably worse than the steady-state numbers we report.

Reliance on verifiable outcomes. The asynchronous update labels each trace's failure mode through a verified outcome Y. This signal is clean for tasks with structured ground truth (unit-test pass rates, exact match, evidence-supported correctness) but degrades on open-ended tasks such as creative writing or long-form summarization, where the failure-type tree may not refine reliably and $\mathcal{A}_t^{\text{syn}}$ can accumulate workflows whose apparent utility does not reflect true failure recovery.

E PROMPT TEMPLATES

This appendix provides the prompts used by the three model roles in DYNAMAS (cf. Section 4): the Scout small LM $\mathcal{M}_S$, the Judge $\mathcal{M}_L$, and the parallel summarizer $\mathcal{M}_P$. All prompts are zero-shot, return strict JSON, and are reused unchanged across datasets; only the per-task fields (e.g., format-compliance hints, symptom vocabulary Σ, candidate set $\mathcal{A}_t$) are substituted at runtime.

E.1 SCOUT RUBRIC PROMPT ($\mathcal{M}_S$)

The Scout small LM is invoked at every checkpoint with the query, the current intermediate output, and the deterministic rule-based features. Its output is capped at 150 tokens and consists of a bounded JSON object $\mathbf{f}_t^{\text{lm}} = (\phi_{\text{LM}}, \sigma_t)$.

Scout rubric prompt

```
You are a fast quality estimator for a multi-agent reasoning workflow.
Score the current intermediate output and assign a single symptom
↪  label.

[USER QUERY]
{q}

[CURRENT INTERMEDIATE OUTPUT]
{o_t}

[RULE-BASED SIGNALS]
- local_consistency  (phi_cons)   : {phi_cons}    # in [0,1]
- agent_reliability  (phi_hist)   : {phi_hist}    # min/mean/last
- format_compliance  (phi_format) : {phi_format}  # 0 or 1

[SYMPTOM VOCABULARY Sigma]
{Sigma}     # task-specific list, e.g. RAG or code-generation labels

[INSTRUCTIONS]
1. Estimate phi_LM in [0,1]: the probability that o_t is a
   correct, complete, and well-grounded answer to q.
   - 1.0 = clearly correct and sufficient.
   - 0.5 = plausible but uncertain or partially supported.
   - 0.0 = clearly wrong, unsupported, or unusable.
2. Pick exactly one label sigma from Sigma that best describes the
   dominant problem with o_t. If o_t looks clean, return "none".
3. Be terse. Do not output anything outside the JSON.
4. Total response must not exceed 150 tokens.

[RETURN STRICT JSON]
{"phi_LM": <float in [0,1]>, "sigma": "<label from Sigma or none>"}
```

E.2 JUDGE PROMPT ($\mathcal{M}_L$)

The Judge is invoked only on EARLYEXIT and AUGMENT states. It receives a compressed trace, the Scout's feature vector and class, the inferred top-K failure modes $\hat{\mathcal{T}}_t$ from the online decision tree, and the current candidate set $\mathcal{A}_t$. It returns a strict JSON record naming the most likely failure mode, selecting an action from $\mathcal{A}_t \cup \{\text{STOP}\}$, and emitting a short rationale.

Judge prompt

```
You are the routing controller of a multi-agent workflow.
The Scout has already screened the current state and flagged it as
↪ {c}.
Your job is to make the final routing decision for this checkpoint.

[USER QUERY]
{q}

[COMPRESSED TRACE h_t_tilde]
# step-indexed list of (agent_name, action, short_summary_of_output)
{h_t_tilde}

[CURRENT INTERMEDIATE OUTPUT]
{o_t}

[SCOUT SIGNALS f_t]
- phi_cons   : {phi_cons}
- phi_hist   : {phi_hist}      # min / mean / last
- phi_format : {phi_format}
- phi_LM     : {phi_LM}
- sigma      : {sigma}
- aggregate utility u_t : {u_t}

[SCOUT CLASS]
c = {c}    # one of EARLYEXIT, AUGMENT

[TOP-K LIKELY FAILURE MODES (from online tree M)]
# pairs of (failure_label, posterior_probability)
{K_t}

[CANDIDATE WORKFLOW SET A_t]
# each entry: (workflow_id, short_description, targets_failure_modes)
{A_t}

[DECISION RULES]
- If c == EARLYEXIT:
    * Default action is STOP.
    * Choose a workflow from A_t only if you see a concrete
      residual concern (e.g., fluent answer with unverified
      citation, format-passing code with untested edge case).
    * In that case, set "failure_mode" to the residual concern.
- If c == AUGMENT:
    * "failure_mode" MUST be drawn from K_t.
    * "action" MUST be a workflow_id from A_t whose
      "targets_failure_modes" includes the chosen failure_mode.
    * Prefer the lowest-cost workflow that addresses the failure.
- Rationale: at most 2 sentences, grounded in the signals above.
- Do not invent workflow ids or failure labels.

[RETURN STRICT JSON]
{
  "failure_mode": "<label from K_t, residual concern, or none>",
  "action":       "<workflow_id from A_t, or STOP>",
  "rationale":    "<= 2 sentences>"
}
```

E.3 PARALLEL SUMMARIZER PROMPTS ($\mathcal{M}_P$)

The summarizer runs asynchronously off the critical inference path (Algorithm 2). It is invoked in two distinct roles: (i) labeling the failure type of a completed trace, which feeds the online update of the decision tree M; and (ii) synthesizing a new candidate workflow when the diagnosed failure type is not addressed by any $a \in \mathcal{A}_{T-1}$.

E.3.1 FAILURE LABELING: $\mathcal{M}_P^{\text{summary}}$

Summarizer prompt — failure labeling

```
You are a post-hoc failure analyzer for a completed multi-agent run.
Given the full trace and the verified outcome, identify the single
failure mode that best explains why this run produced an incorrect or
insufficient answer.

[USER QUERY]
{q}

[FULL TRACE h_T]
# ordered list of (step, agent_name, action, output)
{h_T}

[FINAL OUTPUT o_T]
{o_T}

[VERIFIED OUTCOME Y]
# task-specific ground-truth signal:
#   - code: unit-test pass/fail and failing test names
#   - math: exact-match against gold
#   - RAG : evidence-supported correctness flag and gold answer
{Y}

[EXISTING FAILURE-MODE INVENTORY T]
{T}

[INSTRUCTIONS]
1. If Y indicates success, return failure_mode="none".
2. Otherwise, pick the single label from T that best explains the
   failure. Prefer existing labels.
3. Only set is_new=true if no label in T fits. New labels must be
   short (<=4 words), specific, and disjoint from existing entries.
4. evidence_step is the index in h_T at which the failure first
   becomes diagnosable.

[RETURN STRICT JSON]
{
  "failure_mode":  "<label from T, new label, or none>",
  "is_new":        <true|false>,
  "evidence_step": <int>,
  "explanation":   "<= 3 sentences>"
}
```

E.3.2 WORKFLOW SYNTHESIS: $\mathcal{M}_P^{\text{synthesize}}$

Summarizer prompt — workflow synthesis

```
You are a workflow synthesizer for a multi-agent system. A failure has
been diagnosed that no existing workflow in the candidate set
↪ addresses.
Propose one new corrective workflow that specifically remediates this
failure mode.

[FAILED TRACE h_T]
{h_T}

[DIAGNOSED FAILURE MODE tau_T]
{tau_T}

[EXISTING CANDIDATE SET A_{T-1}]
# (workflow_id, short_description, targets_failure_modes)
{A_T_minus_1}

[TASK-LEVEL CONTEXT]
- task_family   : {task_family}     # e.g. code_generation, rag
- agent_pool    : {agent_pool}      # available agent roles and tools
- io_schema     : {io_schema}       # expected input/output of a step

[CONSTRAINTS]
1. Output must follow the workflow code format used by AFlow / ADAS:
   a parseable Python definition with a single entry function
   `run(state) -> state` that takes and returns the workflow state.
2. The workflow MUST specifically target tau_T --- not a generic
   "refine again" loop.
3. Must NOT duplicate any (id or behavior) entry in A_{T-1}.
4. Must execute end-to-end on the provided dry_run input under the
   given agent_pool. No external network calls beyond the listed
   ↪ tools.
5. Keep the new workflow small: at most 3 agent calls.

[RETURN STRICT JSON]
{
  "name":        "<short_snake_case_id>",
  "targets":     "<failure mode it addresses, exactly tau_T>",
  "code":        "<full workflow code as a single Python string>",
  "dry_run_in":  "<one example input that exercises tau_T>",
  "rationale":   "<why this remediates tau_T, <=3 sentences>"
}
```

F FAILURE-MODE VOCABULARY Σ

The Scout's symptom σ_t is drawn from a bounded, task-specific vocabulary Σ. The vocabulary seeds the root of the failure-type mapping M (Section 4.2.1) and the failure inventory $\mathcal{T}$ used by the summarizer. Below we list the seed labels used in our experiments for the two task families covered in the main results.

F.1 CODE GENERATION (HUMANEVAL, MBPP)

Table 12 lists the seed labels used to initialize the failure-mode mapping for code generation. The labels cover failures observable from generated programs and test feedback, allowing the router to associate a diagnosed failure with an appropriate repair workflow.

Label	Description
`syntax_error`	Generated program does not parse or fails an AST check.
`runtime_exception`	Program parses but raises an uncaught exception on the public tests.
`logic_error`	Program runs to completion but returns wrong values on visible tests.
`edge_case_failure`	Passes typical inputs but fails on boundary or empty inputs.
`off_by_one`	Loop or index range is shifted by one (a frequent specialization of `logic_error`).
`spec_violation`	Function signature, return type, or stated I/O contract is not respected.
`wrong_algorithm`	Approach is structurally incorrect (e.g., greedy where DP is required).
`missing_helper`	Code references an undefined function, variable, or import.

Table 12: Seed failure-mode vocabulary for code generation.

Label	Description
`evidence_hallucination`	Answer asserts facts not present in any retrieved passage.
`missing_evidence`	A required supporting document is absent from the retrieved set.
`second_hop_gap`	First-hop retrieval found the bridge entity but the second-hop document was not fetched.
`decomposition_error`	Sub-question split is incomplete, redundant, or misaligned with the multi-hop structure.
`citation_format_error`	Final answer cites passages with malformed or unresolvable identifiers.
`reasoning_failure`	Required passages are present but the answer is not entailed by them.
`entity_disambiguation`	Wrong referent chosen for an ambiguous mention (e.g., two people share a name).
`answer_format_error`	Output violates the task's answer-format constraints (length, span, units).

Table 13: Seed failure-mode vocabulary for RAG.

F.2 Retrieval-Augmented Generation (HotpotQA)

Table 13 defines the seed labels for retrieval-augmented generation. They distinguish failures in evidence acquisition, multi-hop decomposition, grounding, and answer formatting, which require different retrieval or verification responses.

G Candidate Workflow Library $\mathcal{A}_0$

The initial candidate set $\mathcal{A}_0$ is seeded by pilot runs on the development split (Section 4.2.2). The synthesized set $\mathcal{A}_t^{\text{syn}}$ grows online from observed failures diagnosed by $\mathcal{M}_P$. We list the seed library used per task family; each entry exposes a workflow id, a one-line behavior summary, and the failure modes it primarily targets (cf. Tables 12–13).

G.1 Code Generation (HumanEval, MBPP)

Table 14 specifies the initial code-generation workflow library used in pilot runs. Each entry is a candidate executable workflow with explicit target failure modes; online synthesis extends this seed set only when the observed failure is not adequately covered.

G.2 Retrieval-Augmented Generation (HotpotQA)

Table 15 specifies the initial workflow library for multi-hop retrieval and answer grounding. The workflows target complementary stages of the RAG pipeline, including query decomposition, evidence retrieval, reranking, and claim verification.

Workflow id	Behavior	Targets
`self_consistency_vote`	Sample k programs at temperature τ; pick the one passing the most public/visible tests, ties broken by majority output equivalence.	`logic_error`, `off_by_one`
`test_driven_repair`	Run the public unit tests, feed failing test names and stack traces back to a refiner agent for one repair pass.	`runtime_exception`, `logic_error`
`adversarial_test_synth`	Critic agent generates boundary and adversarial tests; refiner repairs against any failures.	`edge_case_failure`, `off_by_one`
`spec_re_read`	Re-prompt the solver with the original spec, the current code, and a checklist of declared constraints.	`spec_violation`, `missing_helper`
`step_decomposition`	Decompose the task into sub-functions, solve each independently, then integrate.	`wrong_algorithm`
`code_review_critic`	Critic agent produces a structured review (correctness, complexity, edge cases); refiner applies a single patch.	`logic_error`, `wrong_algorithm`
`io_tracing_repair`	Trace execution on small concrete inputs, locate the first divergence from expected behavior, regenerate the offending block.	`logic_error`, `off_by_one`

Table 14: Initial workflow library $\mathcal{A}_0$ for code generation.

Workflow id	Behavior	Targets
`question_re_decomposition`	Re-split the original query into sub-questions guided by the current draft answer and any unresolved bridge entities.	`decomposition_error`
`second_hop_retrieval`	Extract the bridge entity from o_t and issue a fresh retrieval query conditioned on it.	`second_hop_gap`, `missing_evidence`
`multi_source_rerank`	Re-rank the candidate passage pool with an ensemble of BM25, TF-IDF, and a dense scorer; keep the top-m.	`missing_evidence`
`evidence_validation`	For each claim in o_t, retrieve the supporting span and require explicit textual entailment before keeping the claim.	`evidence_hallucination`, `reasoning_failure`
`citation_grounding`	Re-attach citations to surviving claims; drop or rewrite claims with no resolvable citation.	`citation_format_error`, `evidence_hallucination`
`hallucination_detection`	Run an NLI checker over (answer, retrieved evidence); on a contradiction, regenerate the answer constrained to entailed spans.	`evidence_hallucination`
`multi_agent_debate`	Two solver agents argue over the answer with access to retrieved evidence; a judge agent picks the better-supported side.	`reasoning_failure`, `entity_disambiguation`
`entity_disambiguation`	Detect ambiguous mentions, expand the query with disambiguating attributes, and re-retrieve.	`entity_disambiguation`

Table 15: Initial workflow library $\mathcal{A}_0$ for RAG.