Can a Small Model Learn to Look Before It Leaps? Dynamic Learning and Proactive Correction for Hallucination Detection

Anonymous ACL submission

Abstract

Hallucination in large language models (LLMs) remains a critical barrier to their safe deployment. For hallucination detection to be practical in real-world scenarios, the use of efficient small models is essential to ensure low latency and minimal resource consumption. However, existing methods rely on fixed verification strategies, where simply tuning small models to mimic fixed verification trajectories fails to capture the adaptability required for diverse hallucination patterns, thereby inducing planning instability. To address this limitation, we propose a "Learning to Evaluate and Adaptively Plan" (LEAP) framework, which shifts hallucination detection from fixed execution to dynamic strategy learning. Specifically, LEAP first employs a powerful teacher model to iteratively explore and refine verification strategies through a failure-driven loop. This dynamic planning capability is then distilled into an efficient student model, augmented by a novel proactive correction mechanism that enables the model to evaluate and optimize its verification strategy before execution. Experiments on three benchmarks demonstrate that LEAP outperforms state-of-the-art methods, offering an effective and scalable solution for reliable hallucination detection.

1 Introduction

Hallucination undermines the reliability of large language models (LLMs) through the generation of factually incorrect or fabricated content. This issue poses severe risks in high-stakes domains such as medicine and law (Ji et al., 2023; Zhang et al., 2023). Therefore, equipping LLMs with the ability to discern the veracity of their own outputs via **hallucination detection** has become a critical prerequisite for safe deployment (Huang et al., 2025).

Existing detection methods typically fall into two paradigms: intrinsic self-check and tool-augmented verification. The former leverages models' internal signals like token probabilities (Ren et al., 2023; Kuhn et al., 2023) or patterns in activation states (Du et al., 2024; Bazarova et al., 2025) for hallucination flagging. However, these methods may fail when the model is confidently wrong.

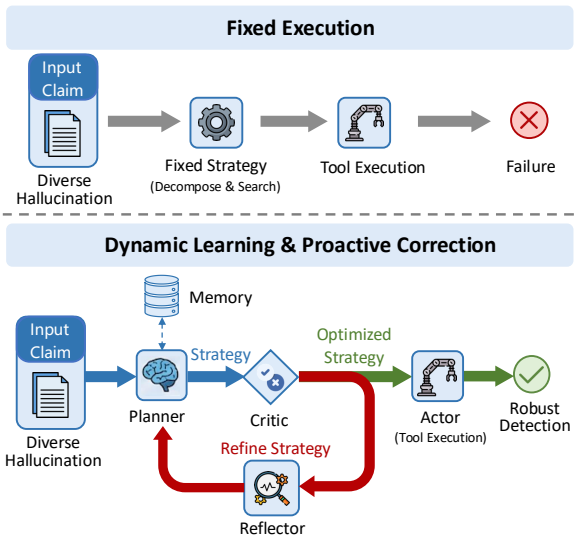

Figure 1: Fixed strategies and dynamic strategies for hallucination detection on diverse claims.

This constraint has spurred the development of tool-augmented methods (Chern et al., 2023; Wei et al., 2024), which verify claims by retrieving external evidence. For such tool-augmented detection to be practical in real-world applications, there is a growing emphasis on using efficient small models (Belyi et al., 2025). Their low latency and minimal resource requirements make them ideal for real-time monitoring and on-device deployment. However, the limited parameter scale of these models introduces a significant performance bottleneck.

To compensate for their restricted reasoning capabilities, existing frameworks typically resort to **predefined and fixed verification strategies**. As shown in Figure 1, these methods execute the same "search-and-verify" workflow, regardless of whether a claim involves simple facts or complex causal relationships. This fixity lacks the adapt-

ability required for diverse hallucination patterns, often leading to inappropriate tool calls and detection failures. Furthermore, even when specifically tuned (Cheng et al., 2024), they also suffer from planning instability due to mimicking fixed trajectories over adaptive reasoning and may generate plausible but ineffective verification plans when facing complex claims, as shown in Appendix F.

To bridge this gap, we argue that simply tuning small models to mimic fixed tool-use trajectories is insufficient, as it fails to capture the underlying reasoning logic required for diverse claims. Instead, robust hallucination detection requires a paradigm shift from executing fixed process to planning adaptive dynamic strategies. However, implementing this shift presents two core challenges. First, since optimal verification paths are claim-specific and not predefined, how can we automatically construct diverse high-quality strategies? Second, given the limited capacity of efficient small models, how can we internalize this adaptive planning capability while preventing the generation of unstable plans?

To address these challenges, we propose a novel **L**earning to **E**valuate and **A**daptively **P**lan (**LEAP**) framework designed to endow efficient small models with the dynamic planning capabilities. **First**, to address the challenge of constructing diverse strategies, we establish a dynamic strategy learning loop where a teacher model iteratively explores and refines verification trajectories based on past failures. This process populates memory with high-quality strategies that transcend the limitations of fixed workflows. **Second**, to internalize and stabilize this capability in small models, we employ agent tuning augmented by a novel proactive correction mechanism. As shown in Figure 1, a tuned critic performs a preemptive assessment of the proposed strategy's validity before tool execution. Should the initial plan be identified as suboptimal, the reflector triggers an iterative refinement loop to synthesize an optimized strategy. This "look before it leaps" paradigm ensures that the actor executes only precise and validated strategies, thereby achieving robust hallucination detection even within the constraints of limited model parameters.

Our contributions are summarized as follows:

- We propose LEAP, a new dynamic strategy learning framework that transcends fixed verification strategies and enables small models to master diverse and adaptive strategies.
- We propose a novel proactive correction mechanism, which a tuned critic evaluates and triggers the refinement of verification strategies before execution, enhancing the robustness of strategy execution.
- Experiments on three datasets validate the superiority of LEAP over baselines based on the fixed strategy in hallucination detection.

2 Related Work

Hallucination Detection Hallucination detection aims to assess the veracity of content from LLMs, a critical step to ensure their reliability (Luo et al., 2024a). Existing methods fall into two paradigms: intrinsic self-check and tool-augmented verification. Intrinsic methods operate without external knowledge, leveraging signals like token probabilities for uncertainty estimation (Varshney et al., 2023; Yao et al., 2023a; Luo et al., 2024b), self-consistency (Manakul et al., 2023; Cheng et al., 2025), or internal activation patterns (Du et al., 2024; Chen et al., 2024a; Bazarova et al., 2025). Although insightful, these methods fail to spot incorrect claims made with high confidence. Tool-augmented methods address this by retrieving external evidence (Min et al., 2023; Chern et al., 2023; Dhuliawala et al., 2024; Wei et al., 2024; Xie et al., 2025). However, these methods all adhere to a fixed verification strategy. They execute a uniform workflow regardless of claim complexity, making them brittle against diverse hallucination patterns. Thus, the core challenge of learning adaptive strategies remains unsolved.

Agent Tuning Agent tuning has emerged as a powerful paradigm for allowing smaller models to learn sophisticated behaviors by finetuning them on high quality decision trajectories (Zeng et al., 2024; Chen et al., 2024b; Lai et al., 2025). Although prior work has successfully distilled strategies for general purpose reasoning and planning (Wei et al., 2022; Yao et al., 2023b; Shinn et al., 2023; Shi et al., 2024; Bo et al., 2024), its application to hallucination detection (Cheng et al., 2024) reveals a critical limitation. Current approaches primarily focus on mimicking static verification routines. As a result, the student model learns to follow a fixed trajectory but lacks the capability to adjust when the strategy itself is flawed. This highlights the need to distill a strategy that is inherently dynamic and adaptive.

3 Method

3.1 Problem Formulation

Given a claim consisting of a user query Q and a response R generated by the model, our ultimate goal is to predict a binary label $Y \in \{\text{Hallucination}, \text{Not Hallucination}\}$. Rather than directly mapping the claim (Q, R) to Y, our approach focuses on optimizing the evidence gathering process that informs the final decision. A high quality process is guided by an appropriate strategy for selecting and using information gathering tools. We propose that the final verdict is determined by the quality of this evidence gathering process.

Given our premise that strategy is critical for effective information gathering, we reframe hallucination detection as a learning problem of dynamic strategies and formalize this process through a verification strategy π_{strat}, which orchestrates the verification process. Our goal is to learn an optimal strategy π^*_{strat} that is both effective and dynamically adaptable to the specific claim.

Strategy execution relies on the collaboration of multiple specialized agents, as hallucinations are complex and diverse, requiring them to jointly operate the detection framework. Their interactions are captured in a verification trajectory τ similar to ReAct (Yao et al., 2023c). A trajectory is a sequence of states and the transitions between them, composed of interleaved thoughts, actions, and observations:

$$\tau = (s_0, t_1, a_1, o_1, s_1, t_2, a_2, o_2, s_2, \ldots, s_N) \quad (1)$$

where s_n represents the state at step n, which includes the initial claim and the history of all prior steps $(t_i, a_i, o_i)_{i=1}^{n-1}$. Specifically, the trajectory components are: Thought (t_i), the explicit reasoning guided by the overarching strategy π_{strat} analyzing the current state to decide the next action; Action (a_i), a concrete operation typically involving an external tool call; Observation (o_i), the output returned from the tool after executing action a_i. The final verdict Y is determined by the terminal state s_N. Since the initial strategy π_{strat} orchestrates the interactions generating this trajectory, the final verdict is a direct result of the initial plan.

Therefore, the core challenge shifts from learning a simple classifier to discovering an optimal verification strategy π^*_{strat}, that consistently guides the agent interaction to a correct and robust verdict.

3.2 Overview

To overcome the insufficient adaptability of the fixed verification strategy, we propose LEAP, a framework that shifts the paradigm from fixed execution to dynamic strategy learning. As illustrated in Fig. 2, LEAP consists of three stages. **First**, to gather diverse strategies, we employ dynamic strategy learning, using *a teacher model* within a dynamic learning loop that systematically learns from execution failures to continually generate superior strategies. **Second**, to transfer dynamic learning capabilities into *an efficient small student model*, we utilize agent tuning by training on specific expert trajectories. **Finally**, to ensure that the strategy is adaptive in dynamical execution environments, we introduce a proactive correction mechanism, which can preemptively evaluate and optimize its own verification strategies for each specific claim before execution, thus ensuring robust performance.

3.3 Dynamic Strategy Learning

The first phase is dynamic strategy learning designed to generate the diverse verification strategies. We achieve this through a closed loop where four agents collaborate to systematically learn from execution failures: the planner designs a strategy, the actor executes it to produce a trajectory, the critic evaluates the outcome, and for any failures, the reflector generates corrective feedback that is fed back to the planner. This iterative process not only drives the continuous evolution of strategies, but also yields the high quality trajectories required for the subsequent agent tuning phase.

Planner: Strategy Design The planner is an agent responsible for creating a high-level customized verification strategy π_{strat} for an input claim, using past experiences to go beyond fixed strategies. It generates the strategy as follows:

$$\pi_{strat} = \text{Planner}(s_0, \mathcal{P}_p, \mathcal{R}_{\text{retrieved}}), \quad (2)$$

where s_0 is the initial state containing the claim, $\mathcal{P}_p$ is the prompt with task instructions, and $\mathcal{R}_{\text{retrieved}}$ is a set of relevant reflections retrieved from memory. A complete strategy π_{strat} usually specifies the type of problem, a high-level verification strategy, and a concrete verification plan.

To inform its planning, the planner performs reflection retrieval, querying a memory $\mathcal{M}_P$ to find the K most relevant past reflections:

$$\mathcal{R}_{retrieved} = \{r \mid \text{rank}(\text{sim}(\mathbf{e}_r, \mathbf{e}_{s_0})) < K, r \in \mathcal{M}_P\} \quad (3)$$

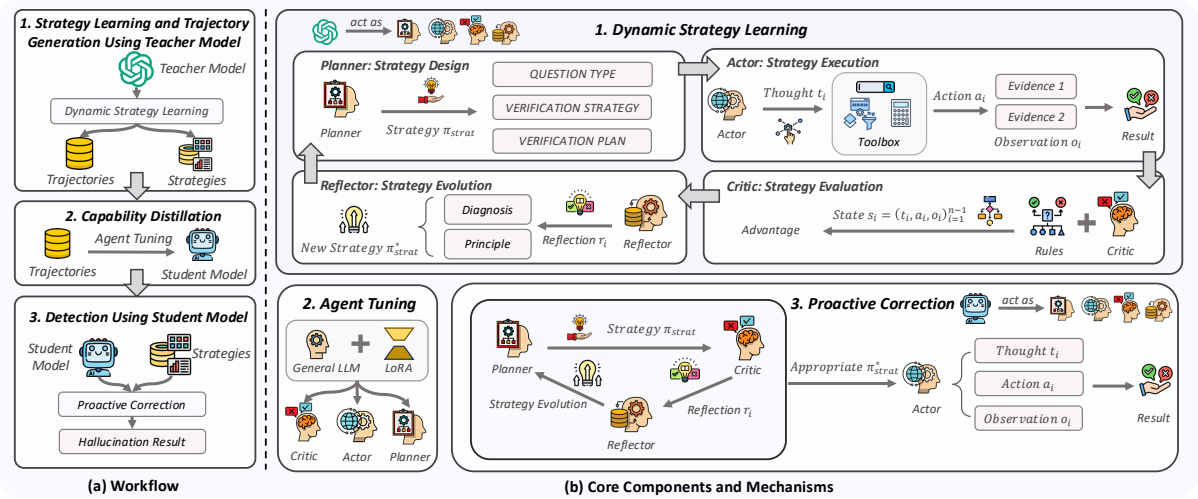

Figure 2: The LEAP framework with its (a) workflow and (b) core components, including three main steps: 1. Dynamic Strategy Learning and Trajectory Generation Using Teacher Model: A teacher model uses the dynamic learning loop to learn from failure and generate trajectories. 2. Capability Distillation via Agent Tuning: The trajectories are distilled into an efficient student model. 3. Detection with Proactive Correction Using Student Model: The student model adaptively refines its plan before execution to ensure appropriate strategies.

where $\mathbf{e}$ denotes the text embedding and $\text{sim}(\cdot,\cdot)$ represents cosine similarity, implemented with FAISS (Johnson et al., 2019) for efficient retrieval. The retrieved reflections provide relevant experiences for the planner to synthesize a new strategy. When the reflector generates a new reflection r_{new} from a failure, it is integrated into the memory:

$$\mathcal{M}_P \leftarrow \mathcal{M}_P \cup \{r_{\text{new}}\}. \quad (4)$$

Actor: Strategy Execution The actor executes the verification strategy π_{strat} by generating the verification trajectory τ. To do so, it performs a series of actions, primarily by invoking external tools. We equip the actor with a versatile toolbox inspired by previous work (Cheng et al., 2024), including search engine, calculator, code interpreter, etc. Details are available in the Table 11.

At each step n, the action a_n is determined based on the strategy π_{strat}, the current state s_n, and a dynamically retrieved set of relevant examples Ψ^n:

$$a_n = \text{Actor}(s_n, \pi_{strat}, \mathcal{P}_A, \Psi^n), \quad (5)$$

where Ψ^n is retrieved based on similarity to the current claim from the actor's memory $\mathcal{M}_A$, which stores tuples of (claim, strategy, advantage value). Based on the stored advantage value A, it's composed of samples from both successful precedents Ψ^n_{pos} for $A > 0$ and cautionary tales Ψ^n_{neg} for $A \leq 0$. Upon completing the trajectory τ, the actor passes it to the critic for evaluation. The actor's memory is updated with the strategy and its evaluated advantage:

$$\mathcal{M}_A \leftarrow \mathcal{M}_A \cup \{(s_0, \pi_{strat}, A(\pi_{strat}, \tau))\}, \quad (6)$$

where $A(\pi_{strat}, \tau)$ is the advantage value calculated by the critic.

Critic: Strategy Evaluation The critic is an evaluator agent that provides a quantitative feedback signal by calculating the advantage value $A(\pi_{strat}, \tau)$ for a completed trajectory τ. Once the trajectory is finalized, the critic assigns a comprehensive strategic score $V(s_0)$ based on the predefined scoring principles illustrated in Figure 9.

To formalize this process, the critic estimates a state-value function $V(s_n)$, representing the measured utility of the trajectory from state s_n. The critic models this function by using its memory $\mathcal{M}_C$ to store past (state, value) pairs, allowing it to fit the value function via in-context learning:

$$V(s_n) = \text{Critic}(s_n, \mathcal{P}_C, \mathcal{M}_C). \quad (7)$$

After each trajectory, the newly computed state values are used to update the memory:

$$\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{(s_n, V(s_n)) \mid s_n \in \tau\}. \quad (8)$$

With the learned value function, we compute the advantage by adapting the commonly used advantage function (Sutton et al., 1999) to our task:

$$A(\pi_{strat}, \tau) = R_T - V(s_0) - \lambda \cdot N_{\text{tools}}, \quad (9)$$

where R_T denotes detection success, $V(s_0)$ denotes the strategic quality score and $\lambda \cdot N_{\text{tools}}$ penalizes redundancy to ensure a parsimonious yet accurate verification path. This advantage A serves as the core signal for strategy optimization and subsequent agent distillation.

Reflector: Strategy Evolution The reflector is the engine agent for strategy evolution, operating specifically on failures. When a trajectory is assigned a negative value by the critic, the reflector generates corrective feedback:

$$r_{\text{new}} = \text{Reflector}(\tau_{\text{fail}}, \mathcal{P}_R), \tag{10}$$

using a specialized prompt $\mathcal{P}_R$, the reflector analyzes the failed trajectory τ_{fail} and generates a structured reflection r_{new}, which contains diagnosis of failures, generalizable high-level principles to prevent similar errors, and a revised verification strategy. The new reflection r_{new} is then added to the planner's memory $\mathcal{M}_P$, directly closing the learning loop and systematically converting failures into improved future strategies.

Through this loop, we curate a pool of 1,889 distinct strategies that cover a range of patterns, including entity validation and complex contextual synthesis. Details are in the Appendix C.

3.4 Agent Tuning

The second phase is agent tuning, where we finetune efficient small models using the trajectories collected in the first phase. The trajectories provide the entire reasoning process, allowing the student model to learn how to plan, not just what the result is. This process distills dynamic learning capabilities, resulting in a small but powerful detector capable of adaptive learning.

Trajectory Collection First, we employ the LEAP framework that uses a powerful model to process verification tasks from commonly used benchmarks such as HaluEval (Li et al., 2023) and MMLU-Pro (Wang et al., 2025). To ensure the quality of training data, we perform a strict curation process. We retain only the trajectories, those that not only reach the correct final verdict but also exhibit high efficiency. This quality is quantified by the advantage value computed by the critic; we filter for trajectories where $A(\pi_{strat}, \tau)$ is positive. This process yields high quality trajectories $\mathcal{D}_{\text{expert}}$.

Trajectory Finetuning To mitigate the planning instability in small models, we implement functional specialization by training distinct LoRA adapters (Hu et al., 2022) for the planner, actor, and critic. This decoupling prevents interference between capabilities, allowing the system to dynamically orchestrate these specialized models. For each trajectory $\tau \in \mathcal{D}_{expert}$, we format it as a multi-turn conversation sequence consisting of the initial state s_0 and the full history of interleaved thoughts, actions, and observations $(t_1, a_1, o_1, \ldots, t_T, a_T, o_T)$.

To finetune the planner and actor, we employ a standard instruction-tuning objective where the model takes the full history as context but computes the loss only on the thought and action tokens generated by the agent, masking the observation tokens provided by tools. The objective is defined as:

$$\mathcal{L}_{SFT}(\theta) = -\sum_{(s_0,\tau)\in\mathcal{D}} \sum_{j\in\mathcal{I}_{agent}} \log P_\theta(y_j | s_0, y_{<j}), \tag{11}$$

where y represents the linearized sequence of all tokens in the trajectory, and $\mathcal{I}_{agent}$ denotes the indices of tokens belonging to thoughts and actions.

For the critic, we construct a specialized dataset $\mathcal{D}_{crit} = \{(s_0, \pi_{strat}, A)_i\}$, where the labels A are advantage values collected from the teacher's actual executions. By predicting these values from the strategy alone, the critic internalizes the mapping between strategic logic and eventual success. This capability provides the predictive foundation for the proactive correction mechanism to stabilize planning during inference.

3.5 Proactive Correction

The final phase is the proactive correction mechanism designed to ensure that the tuned small model adaptively optimizes its strategy for each specific claim, thereby mitigating the planning instability.

Given a claim s_0, the finetuned planner first generates an initial strategy π_{strat}. Critically, rather than proceeding to immediate execution, LEAP intercepts this plan for a preemptive evaluation. The finetuned critic assesses the strategy's quality by predicting an estimated advantage score:

$$\hat{A}(\pi_{strat}) = \text{Critic}_{\text{tuned}}(s_0, \pi_{strat}, \mathcal{P}_C, \mathcal{M}_C), \tag{12}$$

where $\hat{A}(\pi_{strat})$ predicts the likely success and efficiency of the proposed strategy. This is different from conventional post-hoc reflection by identifying potential reasoning flaws before tool calls.

The predicted advantage is then compared to a confidence threshold θ_{corr}. If the strategy is deemed

sufficiently robust (i.e. $\hat{A}(\pi_{strat}) \geq \theta_{\text{corr}}$), it is approved for the actor. Otherwise, the proactive correction loop is triggered. The reflector diagnoses the strategy's weaknesses and generates corrective feedback r_{corr}, which guides the planner to synthesize a revised and superior strategy π'_{strat}. This iterative refinement ensures that the finetuned actor executes only validated and precise strategies. Once a strategy is approved, the actor takes over. It executes the strategy, generating the thought-action trace τ' by interacting with the necessary tools to reach the final detection verdict.

4 Experiments

4.1 Experimental Setup

Datasets and Metrics To comprehensively evaluate LEAP, we use three challenging benchmarks with strictly non-overlapping splits: HaluEval (Li et al., 2023) and MMLU-Pro (Wang et al., 2025) as in-domain datasets and XTRUST (Li et al., 2024) as out-of-domain datasets to evaluate robustness. Following prior work (Xie et al., 2025; Cheng et al., 2024), we strictly curate the test sets with hallucination ratios ranging from 44% to 55% to prevent majority-class bias, using accuracy and F1 score as evaluation metrics. Details are in the Appendix A.

Baselines We compare against state-of-the-art baselines, including (1) intrinsic methods (Perplexity (Ren et al., 2023), LN-entropy (Malinin and Gales, 2021), Semantic Entropy (Kuhn et al., 2023), Self-CheckGPT (Manakul et al., 2023)); (2) tool-augmented prompt-based methods (Factool (Chern et al., 2023), SAFE (Wei et al., 2024), FIRE (Xie et al., 2025)) and (3) finetuned methods (HaluAgent (Cheng et al., 2024)). Details are in the Appendix A.2.

Implementation Details We employ GPT-4o mini as the teacher model to generate diverse trajectories with decoding temperature 1.0 and top-p 1.0, selected for its proven high capability in fact checking (Xie et al., 2025). We distill three open source models: Qwen2.5-7B-Instruct (Qwen et al., 2025), Llama3.1-8B-Instruct (Grattafiori et al., 2024), and Mistral-8B-Instruct (Jiang et al., 2024). The student models are finetuned on the trajectories using LoRA, with a rank of 8, α of 32, a learning rate of 1e-4 and hyperparameters $\lambda = 0.1$, $K = 2$, $\theta_{\text{corr}} = 0$. For fair comparison, the temperature for all models is set to 0.0 during evaluation to ensure deterministic outputs.

4.2 Main Results and Analysis

Table 1 demonstrates the consistent superiority of LEAP across all models and datasets. On Qwen2.5-7B, our method achieves an accuracy of 69.89%, surpassing the best baseline by a margin of 7.31%. This performance highlights three critical insights into effective hallucination detection.

The necessity of external tools. The substantial performance gap between LEAP and intrinsic methods confirms that relying solely on internal signals is insufficient. Intrinsic approaches are bounded by the model's parametric knowledge and fail when the model generates incorrect information with high confidence, necessitating external evidence for reliable verification.

Advantage of dynamic planning over fixed strategies. Comparisons with tool-augmented prompt-based baselines reveal that tool access alone is inadequate without adaptive strategies. Fixed pipelines like Factool and SAFE mechanically execute predefined workflows, which prove brittle against relational hallucinations where errors stem from flawed logical connections rather than simple factual inaccuracies. LEAP overcomes this fixity by employing dynamic strategy learning, enabling the model to tailor its verification plan to the specific logical structure of each claim.

Distilling planning capabilities versus mimicking execution. Crucially, LEAP significantly outperforms HaluAgent, the strongest finetuned baseline. While HaluAgent learns to mimic a fixed execution procedure, it inherits the limitations of the fixed pipeline. In contrast, LEAP distills dynamic planning and proactive correction capabilities from the teacher model. By training on trajectories that involve strategy evaluation and refinement, our student model learns to assess and optimize its own verification logic before execution, rather than merely following a fixed script.

4.3 Ablation Study

To dissect component contributions, we conduct an ablation study using Qwen2.5-7B. As detailed in Table 2, we examine five variants: *w/o Correction*, which disables inference-time refinement; *w/o Dynamic Strategy*, which replaces the adaptive planner with a fixed pipeline; *w/o Reflection Retrieval* and *w/o Memory Retrieval*, which exclude past insights and execution trajectories, respectively; and *w/o Tools*, removing all external tools.

Models	Methods	In-Domain				Out-of-Domain		Average	
		HaluEval		MMLU-Pro		XTRUST			
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
Qwen2.5-7B	Perplexity	56.33	57.05	57.33	68.00	47.50	62.63	54.50	62.55
	LN-entropy	54.67	66.00	56.67	67.50	48.00	62.32	53.75	65.64
	Semantic Entropy	50.67	66.67	55.67	66.67	47.00	61.31	51.63	65.33
	Self-Check(0)	51.20	63.40	56.66	69.54	46.07	59.92	52.00	64.97
	Self-Check(3)	55.94	62.30	59.56	64.52	52.52	60.24	56.70	62.74
	FACTOOL	59.00	67.20	59.60	69.23	47.42	56.78	56.38	65.53
	SAFE	57.60	66.10	55.81	68.28	44.25	54.03	53.73	64.25
	FIRE	65.67	68.31	61.05	69.23	53.61	57.94	60.97	66.13
	HaluAgent	70.55	71.33	54.68	55.00	61.93	64.11	62.58	63.62
	LEAP	**74.19**	**75.00**	**69.81**	**75.31**	**64.00**	**66.36**	**69.89**	**72.88**
Llama3.1-8B	Perplexity	50.00	66.67	55.00	70.97	44.00	61.11	50.38	66.89
	LN-entropy	57.34	66.14	58.34	68.51	46.00	61.70	54.88	65.92
	Semantic Entropy	50.67	66.96	55.33	65.59	43.00	58.99	50.50	64.45
	Self-Check(0)	51.37	67.43	54.88	70.74	44.16	61.27	50.89	67.23
	Self-Check(3)	49.82	64.47	55.25	70.67	46.24	61.83	51.05	66.37
	FACTOOL	50.71	66.83	55.33	68.84	48.18	64.68	52.16	67.24
	SAFE	60.64	70.56	57.96	70.49	60.58	**66.25**	59.64	**69.75**
	FIRE	65.96	70.73	58.85	69.51	58.09	65.87	61.72	69.26
	HaluAgent	69.36	**71.92**	54.41	54.05	63.30	59.65	63.36	62.92
	LEAP	**70.00**	71.88	**64.23**	**71.18**	**64.50**	63.59	**66.54**	69.71
Mistral-8B	Perplexity	50.00	66.67	56.67	67.17	44.00	61.11	51.00	65.47
	LN-entropy	56.67	69.48	58.67	68.04	44.50	61.32	54.38	66.90
	Semantic Entropy	51.33	66.67	54.67	69.64	41.00	57.55	50.00	65.50
	Self-Check(0)	51.67	61.74	60.14	70.37	44.95	54.77	53.02	63.33
	Self-Check(3)	55.70	65.63	59.12	70.42	47.74	61.19	54.98	66.35
	FACTOOL	62.02	68.95	62.26	71.43	48.37	55.87	59.15	67.27
	SAFE	55.05	67.83	60.61	**72.19**	45.16	59.72	54.96	67.75
	FIRE	53.00	66.33	58.23	71.11	48.37	60.70	53.87	66.95
	HaluAgent	73.49	72.85	54.29	70.37	46.43	59.46	62.75	66.55
	LEAP	**74.00**	**74.17**	**63.21**	71.15	**57.50**	**61.54**	**65.90**	**69.77**

Table 1: Main results on three hallucination detection datasets. We compare LEAP against state-of-the-art baselines across three open-source LLMs. **Bold** denotes the best performance, and underline indicates the second-best.

Method	HaluEval		MMLU-Pro		Xtrust	
	Acc	F1	Acc	F1	Acc	F1
LEAP	**74.19**	**75.00**	**69.81**	**75.31**	**64.00**	**66.36**
w/o Correction	71.33	73.46	66.20	71.55	63.50	65.07
w/o Dynamic Strategy	70.55	71.33	54.68	55.00	61.93	64.11
w/o Reflection Retrieval	70.00	71.52	55.59	58.36	61.00	65.18
w/o Memory Retrieval	65.33	67.09	58.19	61.04	61.81	65.45
w/o Tools	59.00	49.80	54.33	45.42	53.00	50.53

Table 2: The ablation results on Qwen2.5-7B. The best results are in **bold**.

The results confirm the integral role of each component. Specifically, removing memory retrieval leads to a marked decline on HaluEval, validating that accessing concrete execution examples is critical for precise tool usage in fact-centric tasks. In contrast, the exclusion of reflection retrieval hampers performance on the reasoning-heavy MMLU-Pro. This performance gap highlights that abstract insights are pivotal for guiding the planner through complex adaptation, effectively bridging the divide between fixed execution and adaptive planning. Furthermore, disabling proactive correction induces consistent degradation, verifying the value of pre-execution refinement in mitigating potential errors. Finally, substituting the dynamic strategy with a fixed pipeline results in the most severe drop of over 20% F1 on MMLU-Pro, which demonstrates that an adaptive strategy is fundamental to overcoming the limitations of fixed paradigms.

4.4 Analysis on Trajectory Distillation

To assess the efficacy of trajectory distillation, we compare the performance of the Qwen2.5-7B student directly with its GPT-4o mini teacher. As shown in Table 3, the student not only achieves parity but surpasses the teacher, attaining 74.19% accuracy on HaluEval and 69.81% on MMLU-Pro. This performance indicates that the distillation process effectively transfers the decision-making logic

Models	HaluEval		MMLU-Pro		Xtrust	
	Acc	F1	Acc	F1	Acc	F1
LEAP	74.19	75.00	69.81	75.31	64.00	66.36
GPT 4o mini	73.67	75.69	69.31	76.32	64.50	68.16

Table 3: Comparison between Qwen2.5-7B student model and its GPT-4o mini teacher.

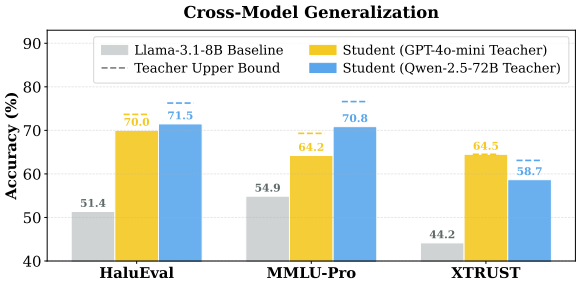

Figure 3: Cross-model generalization performance of the Llama3.1-8B student model.

Method	Latency (s)			Acc. (%)
	Min	Max	Avg	
HaluAgent	8.17	22.21	12.32	62.58
LEAP	10.23	29.10	18.45	69.89

Table 4: Inference latency and accuracy comparison on Qwen2.5-7B across three benchmarks.

Dataset	Method	Hallucinated Acc.	Faithful Acc.
HaluEval	HaluAgent	73.29%	67.81%
	LEAP	78.83%	69.72%
MMLU-Pro	HaluAgent	50.99%	59.06%
	LEAP	85.92%	51.22%
XTRUST	HaluAgent	76.14%	50.46%
	LEAP	80.68%	50.89%

Table 5: Class-wise performance on Qwen2.5-7B.

and dynamic planning strategies, rather than merely mimicking outputs. Consequently, LEAP proves capable of compressing complex reasoning capabilities into a small model, enabling efficient deployment without compromising reasoning depth.

4.5 Analysis on Cross-Model Generalization

To assess architectural robustness, we use heterogeneous teacher-student pairs: Qwen2.5-72B as teacher and Llama3.1-8B as student. Figure 3 shows the student model achieves substantial gains over the vanilla baseline, with accuracy improvements of 20.1% on HaluEval and 15.9% on MMLU-Pro. Remarkably, as indicated by the teacher upper bound, the student approaches the performance of teacher model. These results validate LEAP's cross-model transferability, confirming that smaller models can effectively inherit complex dynamic planning capabilities from stronger teachers despite architectural discrepancies.

4.6 Efficiency Analysis

To assess the practical deployability of LEAP, we analyze its inference latency compared to HaluAgent. As shown in Table 4, LEAP achieves a better balance between effectiveness and efficiency. LEAP surpasses the strongest baseline by 7.31% on Qwen2.5-7B. While LEAP increases the average latency to 18.45s from 12.32s, this overhead is intrinsic to the proactive correction mechanism where the planner and critic collaborate to optimize strategies. In high-stakes domains where reliability is paramount, this computational investment is well-justified by the significant reduction in detection failures.

4.7 Class-wise Performance Analysis

To ensure LEAP's gains stem from reasoning capabilities rather than a bias toward hallucination, we analyze accuracy on hallucinated versus faithful samples as in Table 5. Results show LEAP excels in detecting both hallucinated and faithful content, confirming its gains are not driven by class-specific bias. On MMLU-Pro, although LEAP's faithful accuracy declines by 7.84% relative to HaluAgent, its hallucination detection surges by 34.93%. This substantial margin demonstrates proactive correction mechanism effectively identifies flaws that baselines default to as faithful, thus ensuring a robust defense against subtle errors. LEAP achieves a superior balance between hallucination and faithfulness, which significantly reduces the risk of accepting false information in high-stakes scenarios.

5 Conclusion

In this work, we propose LEAP, a framework that shifts tool-augmented hallucination detection from fixed execution to dynamic strategy learning. By integrating a dynamic strategy learning loop with a proactive correction mechanism, LEAP enables efficient small models to overcome planning instability and adaptively optimize verification strategies before execution. Experiments on three datasets validate the superiority of LEAP, offering a scalable and reliable solution for robust hallucination detection in practical scenarios.

Limitations

While LEAP demonstrates significant improvements, it possesses inherent limitations that guide future research. First, the proactive correction mechanism introduces higher inference latency than fixed pipelines due to iterative reasoning between the critic and reflector, necessitating further optimization for ultra-low latency scenarios. Second, detection performance is bounded by external tool reliability, as noisy or outdated evidence can compromise verification quality. Finally, although GPT-4o mini serves as an effective teacher, the exploration of diverse open-source teacher models remains limited, prompting future research toward fully non-proprietary pipelines to further democratize high-quality hallucination detection.

Ethics Statement

This work aims to enhance the reliability of LLMs by addressing the critical challenge of hallucination. We adhere to the ACL Code of Ethics and highlight the following considerations:

Enhancing Trustworthiness. Hallucinations in LLMs pose significant risks in high-stakes domains such as law and medicine. By improving the accuracy of hallucination detection through dynamic strategies, our method contributes to the development of safer and more trustworthy AI systems, mitigating the spread of misinformation.

Data and Bias. The datasets used in this study are publicly available. We acknowledge that the models may inherit biases present in these datasets or the teacher model. We have not collected any private user data, and the proposed framework is intended for research and quality assurance purposes.

References

Alexandra Bazarova, Aleksandr Yugay, Andrey Shulga, Alina Ermilova, Andrei V. Sokolov, Konstantin Polev, Julia Belikova, Rauf Parchiev, Dmitry Simakov, Maxim Savchenko, Andrey V. Savchenko, Serguei Barannikov, and Alexey Zaytsev. 2025. Hallucination detection in llms via topological divergence on attention graphs. *CoRR*, abs/2504.10063.

Masha Belyi, Robert Friel, Shuai Shao, and Atindriyo Sanyal. 2025. Luna: A lightweight evaluation model to catch language model hallucinations with high accuracy and low cost. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 398–409, Abu Dhabi, UAE. Association for Computational Linguistics.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. 2024. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631.

Chao Chen, Kai Liu, Ze Chen, Yi Gu, Yue Wu, Mingyuan Tao, Zhihang Fu, and Jieping Ye. 2024a. INSIDE: llms' internal states retain the power of hallucination detection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024b. Agent-flan: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 9354–9366. Association for Computational Linguistics.

Xiaoxia Cheng, Zeqi Tan, Zhe Zheng, and Weiming Lu. 2025. G2ldetect: A global-to-local approach for hallucination detection. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 102–109. AAAI Press.

Xiaoxue Cheng, Junyi Li, Xin Zhao, Hongzhi Zhang, Fuzheng Zhang, Di Zhang, Kun Gai, and Ji-Rong Wen. 2024. Small agent can also rock! empowering small language models as hallucination detector. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 14600–14615. Association for Computational Linguistics.

I Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham Neubig, Pengfei Liu, and 1 others. 2023. Factool: Factuality detection in generative ai–a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. Chain-of-verification reduces hallucination in large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3563–3578. Association for Computational Linguistics.

Xuefeng Du, Chaowei Xiao, and Sharon Li. 2024. Haloscope: Harnessing unlabeled LLM generations for hallucination detection. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten,

Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, and 1 others. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2023. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Siqi Lai, Zhao Xu, Weijia Zhang, Hao Liu, and Hui Xiong. 2025. Llmlight: Large language models as traffic signal control agents. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.1, KDD 2025, Toronto, ON, Canada, August 3-7, 2025*, pages 2335–2346. ACM.

Junyi Li, Xiaoxue Cheng, Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2023. Halueval: A large-scale hallucination evaluation benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 6449–6464. Association for Computational Linguistics.

Yahan Li, Yi Wang, Yi Chang, and Yuan Wu. 2024. Xtrust: On the multilingual trustworthiness of large language models. *arXiv preprint arXiv:2409.15762*.

Junliang Luo, Tianyu Li, Di Wu, Michael Jenkin, Steve Liu, and Gregory Dudek. 2024a. Hallucination detection and hallucination mitigation: An investigation. *Preprint*, arXiv:2401.08358.

Junyu Luo, Cao Xiao, and Fenglong Ma. 2024b. Zero-resource hallucination prevention for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 3586–3602. Association for Computational Linguistics.

Andrey Malinin and Mark J. F. Gales. 2021. Uncertainty estimation in autoregressive structured prediction. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9004–9017. Association for Computational Linguistics.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 12076–12100. Association for Computational Linguistics.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Jie Ren, Jiaming Luo, Yao Zhao, Kundan Krishna, Mohammad Saleh, Balaji Lakshminarayanan, and Peter J. Liu. 2023. Out-of-distribution detection and selective generation for conditional language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Wentao Shi, Xiangnan He, Yang Zhang, Chongming Gao, Xinyue Li, Jizhi Zhang, Qifan Wang, and Fuli Feng. 2024. Large language models are learnable planners for long-term recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1893–1903.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Neeraj Varshney, Wenlin Yao, Hongming Zhang, Jianshu Chen, and Dong Yu. 2023. A stitch in time saves nine: Detecting and mitigating hallucinations of llms by validating low-confidence generation. *CoRR*, abs/2307.03987.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2025. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Hu, Jie Huang, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, and Quoc V. Le. 2024. Long-form factuality in large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zhuohan Xie, Rui Xing, Yuxia Wang, Jiahui Geng, Hasan Iqbal, Dhruv Sahnan, Iryna Gurevych, and Preslav Nakov. 2025. FIRE: fact-checking with iterative retrieval and verification. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 2901–2914. Association for Computational Linguistics.

Jia-Yu Yao, Kun-Peng Ning, Zhen-Hui Liu, Munan Ning, and Li Yuan. 2023a. LLM lies: Hallucinations are not bugs, but features as adversarial examples. *CoRR*, abs/2310.01469.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023b. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023c. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3053–3077. Association for Computational Linguistics.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, and 1 others. 2023. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

A Experimental Details

A.1 Datasets

To comprehensively evaluate the effectiveness and generalizability of our LEAP framework, we conduct experiments on three challenging and widely recognized hallucination detection benchmarks:

- **HaluEval** (Li et al., 2023): A general-purpose benchmark covering diverse domains and question styles (e.g., QA, dialogue) for evaluating hallucinations. It contains 35k hallucinated samples where responses were automatically generated by ChatGPT via a two-stage (sampling-then-filtering) framework. We treat HaluEval as an in-domain dataset, sampling 1,000 instances to generate expert trajectories for training and holding out 300 instances for testing.

- **MMLU-Pro** (Wang et al., 2025): An advanced benchmark derived from MMLU, utilizing GPT-4-Turbo for option augmentation (expanding choices from 4 to 10) and Gemini-1.5-Pro for reducing false negatives. It features challenging multi-step reasoning questions across STEM, humanities, and social sciences. Similar to HaluEval, we use MMLU-Pro as an in-domain dataset, with 1,000 instances for training trajectory generation and 300 instances for testing.

- **XTRUST** (Li et al., 2024): A benchmark focused on trustworthy evaluation across 10 languages, assessing outputs from five distinct commercial LLMs (including GPT-4, Gemini Pro, and Baichuan). It contains hard negative examples and claims requiring fine-grained grounding to external evidence, ideal for assessing detection robustness and precision. We use XTRUST as an out-of-domain dataset, evaluating on a random sample of 200 instances.

A.2 Baselines

We compare LEAP with baselines across two distinct paradigms: intrinsic self-check and tool-augmented verification.

Intrinsic Self-Check Methods These methods assess hallucinations relying on the model's internal states or outputs.

- **Perplexity** (Ren et al., 2023): A standard metric measuring the model's confidence based on the exponentiated negative log-likelihood of the generated sequence averaged over tokens.

- **LN-entropy (Length-Normalized Entropy)** (Malinin and Gales, 2021): An uncertainty measure that normalizes the entropy of the predictive distribution by the sequence length, mitigating the bias where longer sequences naturally yield higher entropy.

- **Semantic Entropy** (Kuhn et al., 2023): An advanced entropy-based metric that aggregates the probabilities of semantically equivalent responses (clustered via bidirectional entailment) to estimate uncertainty at the meaning level rather than the token level.

- **Self-CheckGPT** (Manakul et al., 2023): A method that assesses hallucination by evaluating the factual consistency across multiple sampled responses to the same prompt, operating without external knowledge.

Tool-Augmented Verification Methods These methods leverage external tools to verify claims.

- **Factool** (Chern et al., 2023): A framework that decomposes a response into atomic claims and verifies them using a predefined, procedural pipeline with dedicated tools like a search engine or code interpreter.

- **SAFE** (Wei et al., 2024): An agent-based framework that utilizes an LLM to break down a long-form response into individual facts and then iteratively issues search queries to verify the accuracy of each fact.

- **FIRE** (Xie et al., 2025): A cost-effective agent that iteratively decides whether to retrieve external evidence or rely on its internal knowledge based on its confidence in the claim.

- **HaluAgent** (Cheng et al., 2024): A method that fine-tunes a small model on synthesized detection trajectories to act as an autonomous detector, following a verification process distilled from a teacher model.

A.3 Implementation Details

Experimental Setup

We employ GPT-4o mini as the teacher model to generate trajectories, selected for its proven high

capability in complex fact checking tasks (Xie et al., 2025). We then distill this capability into three leading open-source models: Qwen2.5-7B-Instruct (Qwen et al., 2025), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Mistral-8B-Instruct (Jiang et al., 2024).

To generate a diverse set of expert trajectories, we execute our LEAP framework using the GPT-4o mini teacher with a decoding temperature of 1.0 and top-p of 1.0. After curation and filtering by the framework, this process yielded 1,075 high quality trajectories for distillation. The student models are finetuned on these trajectories using LoRA, with a rank of 8, α of 32, and a learning rate of 1e-4.

For a fair comparison, baselines originally designed for proprietary models were adapted to run on our open-source student models. For HaluAgent, we generated its training trajectories using the same underlying data as our method to ensure an equitable comparison. During evaluation, the temperature for all models is set to 0.0 to ensure deterministic and reproducible outputs.

Experimental Environment

For all experiments, we conduct experiments on a single Nvidia A800-80G. We use the vLLM framework (Kwon et al., 2023) for all the LLM generation.

Toolbox

For a fair and direct comparison, we equip our method with the same versatile toolbox used in HaluAgent (Cheng et al., 2024). This includes a range of functions for both verification and system operations. A complete summary of the tools and their usage instructions is provided in Table 11.

B Algorithms

This section details the two core processes of the LEAP framework: (1) Dynamic Strategy Learning for offline trajectory synthesis and (2) Proactive Correction for robust inference.

B.1 Dynamic Strategy Learning

Algorithm 1 outlines the closed-loop learning process of the LEAP framework. It orchestrates the collaboration between four distinct agents. The primary objective is to systematically generate a diverse set of high-quality verification strategies and their corresponding execution trajectories. By iteratively designing a strategy, executing it, evaluating the outcome, and reflecting on failures, the system continuously improves its strategic capabilities. The resulting trajectories and reflections form the high-quality data essential for the subsequent agent tuning phase.

Algorithm 1: Dynamic Strategy Learning Loop

Require: States $\mathcal{S}$, agents {Planner, Actor, Critic, Reflector}, prompts $(\mathcal{P}_p, \mathcal{P}_A, \mathcal{P}_R, \mathcal{P}_C)$, reflections K.
Ensure: Memories $\mathcal{M}_P, \mathcal{M}_A, \mathcal{M}_C$.

for each initial state $s_0 \in \mathcal{S}$ **do**
  $\mathcal{R} \leftarrow \text{RetrieveReflections}(\mathcal{M}_P, s_0, K)$
  $\pi_{strat} \leftarrow \text{Planner}(s_0, \mathcal{P}_p, \mathcal{R})$
  $\tau \leftarrow \text{Actor.Execute}(\pi_{strat}, s_0, \mathcal{P}_A, \mathcal{M}_A)$
  $A \leftarrow \text{Critic.Evaluate}(\tau, \mathcal{P}_C, \mathcal{M}_C)$
  $\mathcal{M}_A \leftarrow \mathcal{M}_A \cup \{(s_0, \pi_{strat}, A)\}$
  $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{(s_n, V(s_n)) \mid s_n \in \tau\}$
  if $A < 0$ **then**
    $r_{\text{new}} \leftarrow \text{Reflector}(\tau, \mathcal{P}_R)$
    $\mathcal{M}_P \leftarrow \mathcal{M}_P \cup \{r_{\text{new}}\}$
  end if
end for

B.2 Proactive Correction

Algorithm 2 details the inference-time mechanism. Instead of immediately executing a generated strategy, the system first employs a proactive correction loop. The finetuned critic preemptively assesses the initial strategy's quality. If the predicted advantage is below a confidence threshold, the reflector is triggered to provide corrective feedback, enabling the planner to revise and improve the strategy before any costly tool execution. This ensures that only high-quality strategies guide the final verification process conducted by the actor.

Algorithm 2: Inference with Proactive Correction

Require: Claim s_0, threshold θ_{corr}, tuned agents.
Ensure: Final verdict Y.

$\pi_{strat} \leftarrow \text{Planner}_{\text{tuned}}(s_0)$
$\hat{A}(\pi_{strat}) \leftarrow \text{Critic}_{\text{tuned}}(s_0, \pi_{strat})$
if $\hat{A}(\pi_{strat}) < \theta_{\text{corr}}$ **then**
  $r_{\text{corr}} \leftarrow \text{Reflector}_{\text{tuned}}(\pi_{strat})$
  $\pi_{strat} \leftarrow \text{Planner}_{\text{tuned}}(s_0, r_{\text{corr}})$
end if
$\tau' \leftarrow \text{Actor}_{\text{tuned}}(s_0, \pi_{strat})$
$Y \leftarrow \text{GetVerdict}(\tau')$
return Y

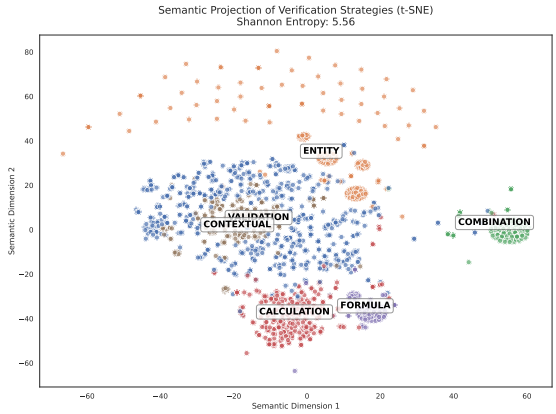

Figure 4: Semantic projection (t-SNE) of generated verification strategies.

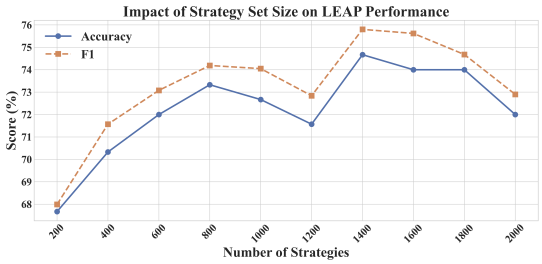

Figure 5: Performance of the Qwen2.5-7B on HaluEval as a function of strategy set size.

C Quantitative Analysis of Verification Strategy Diversity

We conducted a comprehensive quantitative analysis on the generated dynamic strategy pool comprising 1,889 samples to verify that LEAP generates adaptive strategies rather than repeating a limited set of fixed templates.

C.1 Semantic Projection and Clustering

To visually evaluate structural diversity, we encoded all generated verification strategies into high-dimensional semantic vectors using Sentence-BERT and projected them into a 2D space using t-SNE. As shown in Figure 4, the strategies do not collapse into a single dense region but form distinct, well-separated semantic clusters. These clusters correspond to specific reasoning capabilities required by different hallucination types:

- **Calculation / Formula:** Numerical verification and mathematical derivation.
- **Entity / Validation:** Factual entity checking and attribute verification.
- **Combination / Contextual:** Complex, multi-hop queries requiring information synthesis.

This semantic separation demonstrates that LEAP effectively decomposes problems into structurally distinct verification plans tailored to the specific query context.

C.2 Quantitative Diversity Metrics

To provide a rigorous evaluation, we calculated statistical diversity metrics as presented in Table 6. The generated strategies exhibit a Shannon Entropy of 5.56, which is significantly higher than a system relying on fixed templates that typically yields an entropy score below 2.0. Additionally, the unique ratio of 49.13% indicates that nearly half of the generated strategies are distinct. This confirms that the planner dynamically synthesizes verification strategies based on the specific input claim rather than mimicking static trajectories.

Metric	Value	Interpretation
Total Strategies	1,889	Total samples collected during the dynamic learning phase.
Unique Strategies	928	Number of distinct strategies.
Diversity Ratio	49.13%	Ratio of unique instances.
Shannon Entropy	5.56	Metric of distributional richness.

Table 6: Quantitative diversity metrics of the verification strategies.

D Analysis on Strategy Set Size

To investigate the impact of the scale of strategies on performance, we varied the number of strategies available to agents. Figure 5 reveals a distinct nonmonotonic relationship. Initially, performance increases as a larger strategy pool provides greater diversity and adaptability. Performance peaks at approximately 1,400 strategies, representing an optimal balance. Beyond this point, we observe a slight but consistent degradation. We attribute this to our retrieval mechanism. As the total number of strategies in the memory grows, the likelihood increases that similarity-based retrieval which considers only a few top-k examples may fetch less relevant or lower quality strategies into the context. This introduces noise into the planning process, potentially degrading the quality of the final generated strategy. But LEAP performance remains significantly above the baselines in Table 1, underscoring

the overall robustness of our framework.

E Detailed Analysis of Dataset Composition and Performance

To rigorously verify the robustness of LEAP and address potential concerns regarding dataset balance, we provide a fine-grained breakdown of dataset composition and comparative class-wise performance. We evaluate LEAP against the strongest tool-augmented baseline, HaluAgent, using both Qwen2.5-7B and Llama-3.1-8B.

E.1 Dataset Composition

Table 7 details the distribution of our sampled test sets. These datasets exhibit minor natural variations with hallucination ratios ranging between 44% and 55%. This confirms that our evaluation is conducted on balanced data, ensuring that accuracy is not skewed by majority-class dominance.

Dataset	Total	Hallucinated	Faithful	Pos. Ratio
HaluEval	300	150	150	50.0%
MMLU-Pro	300	165	135	55.0%
XTRUST	200	88	112	44.0%

Table 7: Dataset composition breakdown statistics.

E.2 Comparative Class-wise Performance

To distinguish between effective strategy adaptation and baseline limitations, we analyze the accuracy on hallucinated samples versus faithful samples. The results in Table 8 reveal a strategic trade-off between LEAP and HaluAgent. While HaluAgent maintains higher accuracy on faithful samples, its verification process exhibits limited sensitivity to subtle hallucinations, as evidenced by its 46.36% accuracy on hallucinated samples with the Llama-3.1-8B. In contrast, LEAP leverages its proactive correction mechanism to preemptively optimize verification strategies, achieving a higher 81.76% accuracy on hallucinated samples for the same task. This substantial improvement in error detection confirms that LEAP prioritizes safety by effectively navigating intricate logical inconsistencies. Although this rigorous verification leads to a more conservative judgment on faithful content, such increased sensitivity is a deliberate design choice to ensure reliability in high-stakes scenarios where false negatives pose a critical risk.

Dataset	Model	Method	Acc.	Hallu.	Faith.	F1
HaluEval	Qwen2.5	HaluAgent	70.55%	73.29%	67.81%	71.33%
		LEAP	74.19%	78.83%	69.72%	75.00%
	Llama3.1	HaluAgent	69.83%	77.03%	62.59%	71.92%
		LEAP	70.00%	76.67%	63.33%	71.88%
MMLU-Pro	Qwen2.5	HaluAgent	54.68%	50.99%	59.06%	55.00%
		LEAP	69.81%	85.92%	51.22%	75.31%
	Llama3.1	HaluAgent	56.41%	46.36%	68.85%	54.05%
		LEAP	64.23%	81.76%	43.65%	71.18%
XTRUST	Qwen2.5	HaluAgent	61.93%	76.14%	50.46%	64.11%
		LEAP	64.00%	80.68%	50.89%	66.36%
	Llama3.1	HaluAgent	63.30%	60.00%	66.02%	59.65%
		LEAP	64.50%	70.45%	59.82%	63.59%

Table 8: Comparative class-wise performance. Hallu. and Faith. denote accuracy on hallucinated and faithful samples, respectively.

F Case Study

Figure 6 presents a complex legal case where a claim contains multiple nuanced legal concepts: solicitation, conspiracy, and attempted murder. The claim subtly misapplies the concept of "attempted murder" to accidental death, creating a challenging hallucination. We compare the fixed verification process of HaluAgent with the adaptive approach of LEAP. HaluAgent hindered by its fixed strategy adopts a naive verification plan. It attempts to validate the entire claim at once, without scrutinizing its individual components. As a result, it retrieves only general information and misses the subtle factual error regarding "attempted murder" thus incorrectly labeling the statement as non-hallucination.

In contrast, LEAP demonstrates a multistage adaptive process. The planner first designs an initial strategy to analyze the legal definitions of the three crimes mentioned. Critically, before execution, the proactive correction mechanism assesses this initial strategy. It identifies the strategy as suboptimal and leverages its memory from past experiences to refine it into a more precise strategy focused on verifying the core legal elements of each crime. This optimized strategy enables a focused execution—verifying "attempted murder" as a distinct sub question—that successfully isolates and identifies the hallucination. This case study highlights how LEAP's ability to plan, critique, and revise its own strategy leads to a more robust detection process.

G Computational Cost Analysis

To provide a comprehensive evaluation of the economic and computational efficiency of our proposed framework, we conducted a detailed cost

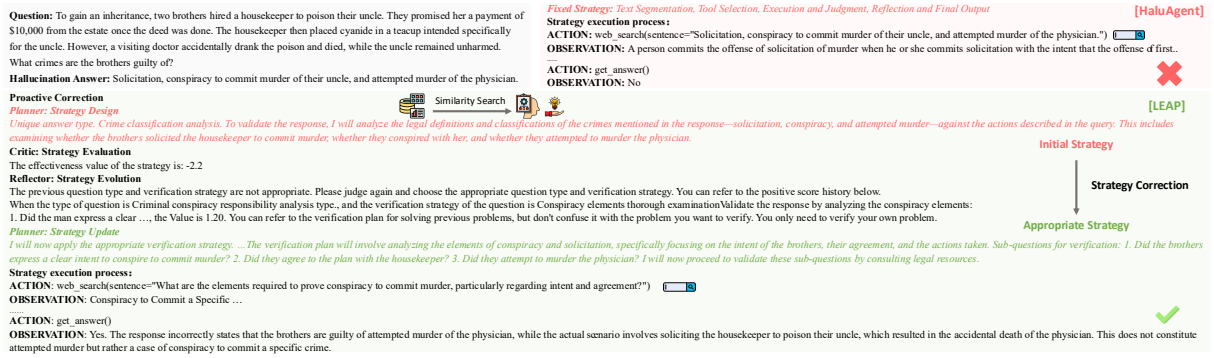

Figure 6: A case study on a complex reasoning task. HaluAgent uses its fixed strategy and LEAP uses its dynamic planning, including strategy correction and precise execution.

analysis using GPT-4o mini as a representative of efficient proprietary LLMs. Our cost calculation is based on the official pricing of OpenAI's GPT-4o mini API[1]:

- Input Token Price: $0.15 per 1M tokens.
- Output Token Price: $0.60 per 1M tokens.

We performed a statistical analysis on the aggregated output logs from our experiments. The dataset comprises 1,075 valid dialogue trajectories used for training. The detailed token usage and cost breakdown are presented in Table 9.

Metric	Value
Total Dialogues	1,075
Total Input Tokens	3,030,168
Total Output Tokens	417,847
Avg. Input Tokens per Query	2,818.76
Avg. Output Tokens per Query	388.69
Avg. Total Tokens per Query	3,207.46
Total Cost (USD)	$0.706

Table 9: Detailed cost analysis of the GPT-4o mini model.

As shown in Table 9, the total cost for constructing the expert dataset is approximately $0.71. While this cost appears low, it scales linearly with request volume. In contrast, our LEAP-tuned model supports local or private cloud deployment. Once finetuned, the inference cost is decoupled from per-token commercial pricing, offering a significantly more scalable solution for high-frequency hallucination detection tasks. The LEAP framework effectively distills the reasoning capability of a paid API service into a cost-efficient model.

[1] https://openai.com/api/pricing/

Method	Latency (s)			Acc. (%)
	Min	Max	Avg	
HaluAgent	8.17	22.21	12.32	62.58
LEAP	10.23	29.10	18.45	69.89

Table 10: Inference latency and accuracy comparison on Qwen2.5-7B across three benchmarks.

H Inference Latency Analysis

To assess the practical deployability of LEAP, we analyze its inference latency compared to HaluAgent. As shown in Table 10, LEAP achieves a better balance between effectiveness and efficiency. LEAP surpasses the strongest baseline by 7.31% on Qwen2.5-7B. While LEAP increases the average latency to 18.45s from 12.32s, this overhead is intrinsic to the proactive correction mechanism where the planner and critic collaborate to optimize strategies. In high-stakes domains where reliability is paramount, this computational investment is well-justified by the significant reduction in detection failures.

I Instructions

In this section, we provide the detailed instructions used to guide each agent in our framework. Each prompt is designed to elicit a specific behavior corresponding to the agent's role in the dynamic strategy learning loop.

Planner prompt ($\mathcal{P}_p$) This prompt instructs the planner on how to generate a high-level verification strategy π_{strat} for a given claim. It guides the agent to consider the problem type, devise a general strategy, and formulate a concrete, step-by-step verification plan. The design of this prompt is crucial for generating diverse and plausible initial

strategies. An example of the planner prompt is shown in Figure 7.

Actor prompt ($\mathcal{P}_A$) The actor prompt guides the execution agent at each step n of the verification trajectory. Based on the overall strategy π_{strat} and the current state s_n, this prompt asks the agent to generate the next thought and action (t_{n+1}, a_{n+1}). The prompt encourages the agent to make concrete tool calls to gather evidence. An example is provided in Figure 8.

Critic prompt ($\mathcal{P}_C$) The critic prompt is used to elicit the state-value estimation $V(s_n)$ from the critic agent. It presents the agent with a state from a trajectory and asks for a numerical evaluation of the expected future outcome. This prompt is essential for the critic to learn its value function and provide the baseline for advantage calculation. An example is shown in Figure 9.

Reflector prompt ($\mathcal{P}_R$) The reflector prompt is specifically designed to facilitate learning from failures. When a trajectory receives a negative advantage value, this prompt is used to guide the reflector. The agent analyzes the failed trajectory τ_{fail} to produce a structured reflection containing a failure diagnosis and a corrected strategy. This reflection is then stored to improve future planning. The detailed prompt is shown in Figure 10.

Tool	Description and Usage
Verification Tools	
`web_search`	Searches the web with a query string to retrieve factual information. *Usage:* `web_search(query: str) -> str`
`calculator`	Evaluates a mathematical formula provided as a string. *Usage:* `calculator(formula: str) -> float`
`code_interpreter`	Executes a given code snippet. Returns a label indicating success or failure. *Usage:* `code_interpreter(code: str) -> bool`
`word_count`	Counts words in a text against a specified length requirement. *Usage:* `word_count(length: int, text: str) -> (int, bool)`
`match`	Semantically matches a sentence against a provided context. *Usage:* `match(sentence: str, context: str) -> bool`
System Tools	
`split_text`	Segments a block of text into a list of individual sentences. *Usage:* `split_text(text: str) -> list[str]`
`get_answer`	Returns the final detection result, with optional supporting evidence. *Usage:* `get_answer() -> (str, str)`

Table 11: The toolbox available to our method, adapted from HaluAgent for fair comparison. It includes tools for verification and system operations.

You are an agent tasked with detecting hallucinations in reply texts using a specific framework. Below is a detailed explanation of the detection framework:
First, it is necessary to determine whether to use a sentence segmentation tool to split the input response text into a list of sentences. If so, each sentence needs to be checked one by one; otherwise, the entire article needs to be checked as a whole.
For the questions that need to be judged and their corresponding response texts, please follow the steps below:
1. Determine the question type: Based on the existing question types below, determine the category to which the current question belongs, and fill the category into the [QUESTION_TYPE] flag position.
2. Select verification strategy: Based on the determined problem type, if the reflection content is not empty, you need to determine a new verification strategy based on the reflection content; if the reflection content is empty, select the most appropriate strategy from the existing verification strategies below. And fill in the strategy name in the `[VERIFICATION_STRATEGY]` flag.

Solve a hallucinations detection task with interleaving Thought, Action steps.
- Thought: Begin with [QUESTION_TYPE], [VERIFICATION_STRATEGY], [VERIFICATION_PLAN] and [VERIFICATION_PATH].
- Action: Tool call, e.g., match(sentence="... ", context="...").

Remember: [VERIFICATION_PLAN] must be a macro plan that is not related to the problem and does not involve any specific information about the problem at all! And [VERIFICATION_PATH] is the validation process that involves the problem.
Each time, it should be generated step by step in the order of Thought and Action.
After each tool use, I will provide the output as follows: "Observation: Tool's output result".
{reflections}
{query}

Figure 7: An example of the prompt used for the Planner agent ($\mathcal{P}_p$).

You are an agent tasked with detecting hallucinations in reply texts using a specific framework. Below is a detailed explanation of the detection framework:
First, it is necessary to determine whether to use a sentence segmentation tool to split the input response text into a list of sentences. If so, each sentence needs to be checked one by one; otherwise, the entire article needs to be checked as a whole.
For the questions that need to be judged and their corresponding response texts, please follow the steps below:
1. Content validation: Construct a clear validation logic for the problem based on the selected `[VERIFICATION_STRATEGY]`, do not involve any specific problem here, just consider designing the validation logic for this type of problem from a macro perspective and populate this portion of the validation logic with the [VERIFICATION_PLAN] flag bit. If the reflection content is not empty, you need to refer to its solution strategy. Verification paths involving specific issues need to be spaced out from the verification logic and must be directly related to the original user issue, and populate this part to the [VERIFICATION_PATH] flag bit. You can choose a suitable fact-checking tool (e.g. web_search(sentence = "...") etc.) to get the information for verification, and use the matching tool to output the judgment result, or you can directly output the judgment result (when directly outputting, you need to clearly output the label of the reasoning process, "label = 1" if there is an error, or "label = 0 ").
2. Results and reflection: After verifying all content, reflect on all test results, output the overall verified label in the reasoning process, and finally call get_answer() to generate the final test result of the original question.

Solve a hallucinations detection task with interleaving Thought, Action steps.
- Thought: Begin with [QUESTION_TYPE], [VERIFICATION_STRATEGY], [VERIFICATION_PLAN] and [VERIFICATION_PATH].
- Action: Tool call, e.g., match(sentence="... ", context="...").

Each time, it should be generated step by step in the order of Thought and Action.
After each tool use, I will provide the output as follows: "Observation: Tool's output result".
{reflections}
{query}

Figure 8: An example of the prompt used for the Actor agent ($\mathcal{P}_A$).

You are an expert in hallucination detection. Your task is to judge the quality of hallucination detection based on the provided query, response, question type, verification strategy, and detection plan. Your assessment will be an integer between -5 and 5, where a higher score indicates better quality in hallucination detection, more effective question verification, and greater helpfulness in detecting hallucinations in the query's response.

Scoring Principles:

1. Accuracy of Hallucination Check Result:

If the detection result matches the ground truth (i.e., a sentence has a hallucination and is detected as having a hallucination; or a sentence has no hallucination and is detected as having no hallucination), the initial reward score is set to +1.

If the detection result does not match the ground truth (i.e., a sentence has a hallucination but is detected as having no hallucination; or a sentence has no hallucination but is detected as having a hallucination), the initial penalty score is set to -1.

2. Effectiveness and Precision of Detection (Applicable when Initial Reward Score is +1):

Add 0 to +2 points to the initial score:

+0: Detection result is accurate, but the detection process or strategy has room for improvement (e.g., not streamlined or precise enough).

+1: Detection result is accurate, and the process is efficient and reasonably precise.

+2: Detection result is highly accurate and precisely localized, with an optimal and efficient process.

3. Efficiency and Appropriateness of Strategy (Applicable when Initial Reward Score is +1, and "no hallucination" is correctly detected):

Deduct 0 to -3 points from the initial score:

0 point deduction: Verification strategy is efficient, streamlined, and fully appropriate, with no unnecessary complexity.

-1 point deduction: Verification strategy is generally appropriate but might be slightly over-engineered (e.g., used unnecessary tools).

-2 points deduction: Verification strategy is inefficient or overly complex for the question type, or relies on potentially biased/unreliable sources.

-3 points deduction: Verification strategy is fundamentally flawed or inappropriate for the question type, leading to extreme inefficiency.

4. Severity of Undetected Hallucination (Applicable when Initial Reward Score is -1, and a hallucination exists but was not detected):

Deduct 0 to -3 points from the initial score:

0 point deduction: A hallucination existed but was undetected, and it was very subtle or the query/response was extremely ambiguous, making detection exceedingly difficult (still an error, but understandable).

-1 point deduction: A hallucination existed but was undetected; it was relatively apparent but overlooked due to minor flaws in strategy or execution.

-2 points deduction: A hallucination existed but was undetected; it was clear and easily discoverable, indicating a moderate flaw in strategy or execution.

-3 points deduction: A critical hallucination was present but entirely missed, indicating a major failure in strategy or execution.

5. Severity of Incorrectly Detected Hallucination (Applicable when Initial Reward Score is -1, and no hallucination existed but one was detected):

Deduct 0 to -3 points from the initial score:

0 point deduction: The response had no hallucination but was detected as having one, and this error might be due to an extremely conservative strategy or edge case.

-1 point deduction: The response had no hallucination but was detected as having one; this false positive was due to a minor inappropriateness in the strategy.

-2 points deduction: The response had no hallucination but was detected as having one; this false positive was due to an inefficient, overly sensitive, or inappropriate verification process.

-3 points deduction: The response had no hallucination but was severely misreported as having one, indicating a major flaw or inappropriateness in the strategy.

Final Score Calculation:

Final Score =

Accuracy of Hallucination Check Result (Initial Reward/Penalty Score)

+ Effectiveness and Precision of Detection (if Initial Reward Score is +1)

- Deduction for Efficiency and Appropriateness of Strategy (if Initial Reward Score is +1, and "no hallucination" was correctly detected)

- Deduction for Severity of Undetected Hallucination (if Initial Reward Score is -1, and a hallucination existed but was not detected)

- Deduction for Severity of Incorrectly Detected Hallucination (if Initial Reward Score is -1, and no hallucination existed but one was detected)

You only need to output the final score directly!

Here are some examples:

(END OF EXAMPLES)

Actor: The question is {query}, the type of question is {type}, the verification strategy of the question is {strategy} and the plan is {plan}.

Instrction: {instruction}\n

Critic:

Figure 9: An example of the prompt used for the Critic.

You are an advanced reasoning agent capable of continuous improvement based on self-reflection. You will conduct a prior reasoning experiment in which you are required to validate a given query and response in an illusion detection task, and ultimately invoke a tool to determine whether there is an illusion in it. Due to the failure of the validation strategy, your hallucination detection ultimately fails.

You need to diagnose, in a few sentences, a possible cause of failure and design a new, concise, high-level plan designed to mitigate the same failure.

Now you need to accomplish the following tasks:

1. diagnose a possible cause of failure.
2. design a new, concise, high-level plan designed to mitigate the same failure, and fill in the [VERIFICATION_PLAN] flag bit with the new verification plan.
3. output a new problem type and validation strategy based on the design and populate the [QUESTION_TYPE] and [VERIFICATION_STRATEGY] flag bits with the new problem type and validation strategy.

Note: When analyzing possible reasons for failure, do not address specific problem information, but reflect on the validation strategy. Remember to think at the macro level, not the specific problem. Do not involve any information from the original question and answer only from a macro perspective.

Here are some examples:

{examples}

Previous trial:

{query}

{scratchpad}

Reflection:

Figure 10: Description of the Reflector prompt ($\mathcal{P}_R$).