PPA-Plan: Proactive Pitfall Avoidance for Reliable Planning in Long-Context LLM Reasoning

Anonymous ACL submission

Abstract

Large language models struggle with reasoning over long contexts where relevant information is sparsely distributed. Although plan-and-execute frameworks mitigate this by decomposing tasks into planning and execution, their effectiveness is often limited by unreliable plan generation due to dependence on surface-level cues. Consequently, plans may be based on incorrect assumptions, and once a plan is formed, identifying errors and revising it reliably becomes difficult, limiting the effectiveness of reactive refinement. To address this limitation, we propose PPA-Plan, a proactive planning strategy for long-context reasoning that focuses on preventing such failures before plan generation. PPA-Plan identifies potential logical pitfalls and false assumptions, formulates them as negative constraints, and conditions plan generation on explicitly avoiding these constraints. Experiments on long-context QA benchmarks show that executing plans generated by PPA-Plan consistently outperforms existing plan-and-execute methods and direct prompting.

1 Introduction

Large language models (LLMs) have become increasingly strong (Brown et al., 2020; Srivastava et al., 2023), with substantial improvements in both reasoning performance and supported context length, enabling their use across a wide range of applications. Nevertheless, when moving beyond surface-level information retrieval or local understanding to complex, multi-step reasoning over long inputs, significant challenges remain. In such settings, task-relevant evidence is often sparsely distributed across distant spans, mixed with irrelevant details, or presented in positions that bias model attention (Liu et al., 2024). This leads to failure modes such as missed evidence, unsupported assumptions, hallucination, and incorrect answers. Even with standard chain-of-thought prompting (Wei et al., 2022), models frequently struggle to construct coherent reasoning trajectories and fail to arrive at correct answers for complex queries, particularly in long-context scenarios. These observations suggest that effective long-context reasoning requires not only more extensive reasoning traces, but also more deliberate control over how reasoning processes are structured and guided.

To address these challenges, recent work has proposed plan-and-execute frameworks that decompose complex long-context tasks into an explicit planning stage followed by step-by-step execution (Yao et al., 2023; Wang et al., 2023; Sun et al., 2024; Erdogan et al., 2025; Hu et al., 2025). While these approaches often leverage LLMs to generate intermediate plans and have demonstrated improvements over direct generation, the planning stage itself remains a critical bottleneck. This is because existing LLM-based planners typically rely on surface-level textual patterns present in long-context inputs when generating plans (as illustrated in Figure 1), rather than faithfully capturing the underlying logical structure of the instruction or task (Valmeekam et al., 2023; Tang et al., 2023). As a result, generated plans often embed implicit or unsupported assumptions, overlook necessary constraints, or commit prematurely to specific reasoning paths, causing errors to cascade throughout subsequent execution.

To mitigate such issues, prior work has largely adopted reactive refinement strategies that revise plans (Sun et al., 2024). However, this approach is fundamentally limited, as LLMs tend to anchor on their own generated outputs and exhibit strong resistance to revising incorrect assumptions once they are formed (Huang et al., 2023; Xi et al., 2025). Consequently, identifying and correcting errors after plan generation is often more difficult than preventing them in advance at the planning stage.

To equip LLMs with the ability to generate reli-

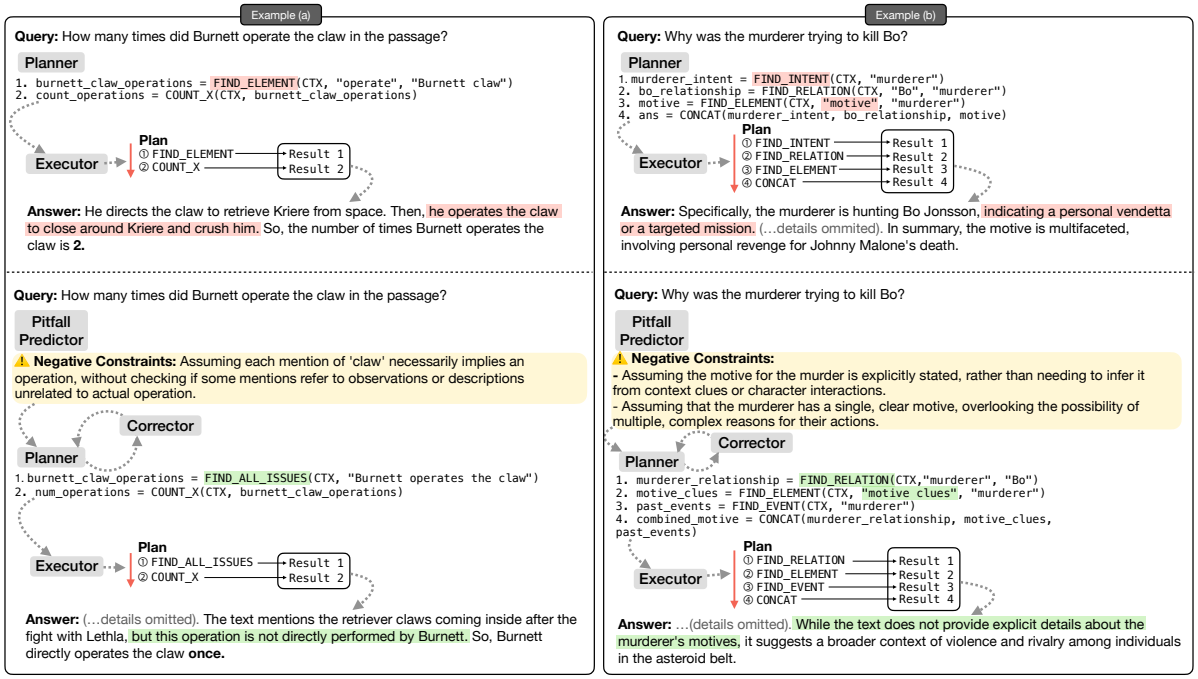


Figure 1: Comparison of planning and execution results by PEARL (Sun et al., 2024) (top) and our PPA-Plan (bottom) on QuALITY (Pang et al., 2022) using GPT-4o-mini. In Example (a), PEARL produces an incorrect answer to a counting query due to surface-level keyword matching that ignores the situational context of who performed each action. In Example (b), PEARL hallucinates a motive for a murder despite the absence of supporting evidence in the document, resulting from premature assumptions. In contrast, PPA-Plan explicitly identifies and avoids such logical pitfalls during plan generation, leading to more reliable reasoning and answers.

able plans to execute them for long-context scenarios, we introduce a planning strategy, PPA-Plan, short for **P**roactive **P**itfall **A**voidance **Plan**ning. PPA-Plan consists of three modules: (1) a *Pitfall Predictor*, (2) a *Planner*, and (3) a *Corrector*. First, the *Pitfall Predictor* analyzes a query to explore potential logical pitfalls and false assumptions, which are formulated as *Negative Constraints* that should be avoided during the plan generation (see Figure 1). Conditioned on these constraints, the *Planner* performs *Strategy Reasoning* to determine the logical structure of a plan while satisfying all the specified negative constraints. After this, if the plan violates syntactic requirements for executability, the *Corrector* is iteratively invoked to repair such errors via *Strategy Reasoning*, while preserving the original intent by referencing both the query and the negative constraint set to preserve the initial plan logic. Once a valid plan is obtained, we sequentially execute its steps to produce an open-ended free-form output using LLMs. From a broader perspective, PPA-Plan can also be viewed as a modular, multi-agent system, where different components specialize in distinct reasoning roles and interact through explicit intermediate outputs (Woolridge, 2001; Yao et al., 2023; Wu et al.; Kim et al., 2024).

We evaluate the effectiveness of PPA-Plan on a subset of long-context question-answering benchmarks, including QuALITY (Pang et al., 2022), ConditionalQA (Sun et al., 2022), and LongReason (Ling et al., 2025), and Qasper (Dasigi et al., 2021), a reading comprehension dataset that contains questions about long-form articles. For multiple-choice question-answering datasets, we map the model's generated free-form answer to one of the candidate options using an LLM and report accuracy, following Sun et al. (2024). Executing plans generated by PPA-Plan consistently yields more accurate answers with improved coverage than existing plan-and-execute strategies or direct prompting baselines, particularly for questions that require reasoning over the full long document.

2 Related Work

Long-Context Reasoning. Recent advancements in LLM architectures and training systems, such as Rotary Position Embedding (RoPE) (Su et al., 2024), Position Interpolation (PI) (Chen et al., 2023), RingAttention (Liu et al., 2023),

YaRN (Peng et al., 2024), and ALiBi (Peng et al., 2024) have significantly enhanced the model's capacity to process extended context lengths; however, the ability to perform complex reasoning over long contexts remains limited, as expanding the context window does not inherently grant the model the capability to synthesize logic across vast information (Liu et al., 2024). This limitation, further evidenced by comprehensive long-context benchmarks like LongBench (Bai et al., 2024a), LooGLE (Li et al., 2024), and RULER (Hsieh et al., 2024), as well as reasoning-intensive tasks in QuALITY (Pang et al., 2022) and LongReason (Ling et al., 2025), motivates our work to focus on enhancing reasoning strategies rather than merely extending input length.

LLM-based Planning. Foundational planning frameworks, such as Plan-and-Solve (Wang et al., 2023) and ReAct (Yao et al., 2023), established the importance of task decomposition; however, they are not inherently designed for long-context scenarios. Subsequent reactive strategies like Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), and ADaPT (Prasad et al., 2024) utilize linguistic feedback to refine logic post-hoc, yet they tend to anchor on initial faulty outputs (Huang et al., 2023; Xi et al., 2025). While PEARL (Sun et al., 2024) successfully integrated these paradigms into long-context tasks, it remains limited by its reliance on reactive refinement and a mechanism designed around surface-level textual patterns. These limitations motivate our focus on proactively preventing logical errors by leveraging an agent-planning framework designed for long-context reasoning.

3 Method

We propose a planning strategy, `PPA-Plan`, that produces reliable execution plans for tasks that require complex reasoning over long documents. The core idea of `PPA-Plan` is to proactively identify potential logical pitfalls and false assumptions before plan generation, and to generate negative constraints that explicitly guide the planning process. By conditioning planning on these constraints, `PPA-Plan` systematically avoids error-prone reasoning paths that are difficult to correct once a plan is formed. `PPA-Plan` consists of three components: (1) a pitfall predictor $\mathcal{M}_{pred}$, (2) a planner $\mathcal{M}_{plan}$, and (3) a corrector $\mathcal{M}_{corr}$. Figure 2 illustrates the overall framework of `PPA-Plan` and a concrete example.

3.1 Pitfall Predictor for Negative Constraint Generation

The *pitfall predictor* $\mathcal{M}_{pred}$ analyzes a query q to identify potential logical pitfalls and false assumptions, which are formalized as a negative constraint set $\mathcal{C}_{neg}$. These constraints specify reasoning patterns that should be avoided during plan generation.

To perform this analysis, $\mathcal{M}_{pred}$ adopts the functional roles of an *exam designer* and a *logic analyst*, critically examining q to uncover implicit premises. Using a structured prompt, as shown in Table 4, with task-specific guidelines and few-shot demonstrations covering multi-hop inference, scope constraints, and counting, the predictor identifies risks such as shallow semantic patterns, scope confusion, and keyword-based guessing.

The predictor derives up to k most critical constraints in a structure JSON format to ensure compatibility with the downstream modules. Formally,

$$\mathcal{C}_{neg} = \{c_1, c_2, \ldots, c_k\} = \mathcal{M}_{pred}(q)$$

The resulting $\mathcal{C}_{neg}$ is passed to the planner to facilitate constraint-aware plan generation.

3.2 Constraint-Aware Plan Generation

Given the query q and negative constraint set $\mathcal{C}_{neg}$, the *planner* $\mathcal{M}_{plan}$ generates an executable plan while avoiding the identified logical pitfalls. Rather than directly producing a plan, $\mathcal{M}_{plan}$ first performs *Strategy Reasoning*, an intermediate reasoning step that determines how the action sequence for the query q should be structured to satisfy the constraints, by analyzing $\mathcal{C}_{neg}$.

This reasoning step is guided by few-shot demonstrations that map the identified pitfalls to concrete strategic adjustments (Table 5). By incorporating exemplars that showcase the causal link between strategy and action selection, the module guides the agent's internal reasoning process toward the intended cognitive path before generating the initial plan. The rationale behind this approach (Strategy Reasoning) is that simply imposing $\mathcal{C}_{neg}$ often leads models to ignore prohibitions or leaves them unsure of alternative actions.

To ensure planning stability and reliable grounding, $\mathcal{M}_{plan}$ selects actions from a pre-defined action space $\mathcal{A}$[1] to translate this derived strategy into

[1]We adopt the standard action space $\mathcal{A}$ and function-call format `output=action(args)` provided by PEARL (Sun et al., 2024). We also allow $\mathcal{M}_{plan}$ to dynamically define and use auxiliary actions if the required operation is absent from $\mathcal{A}$.

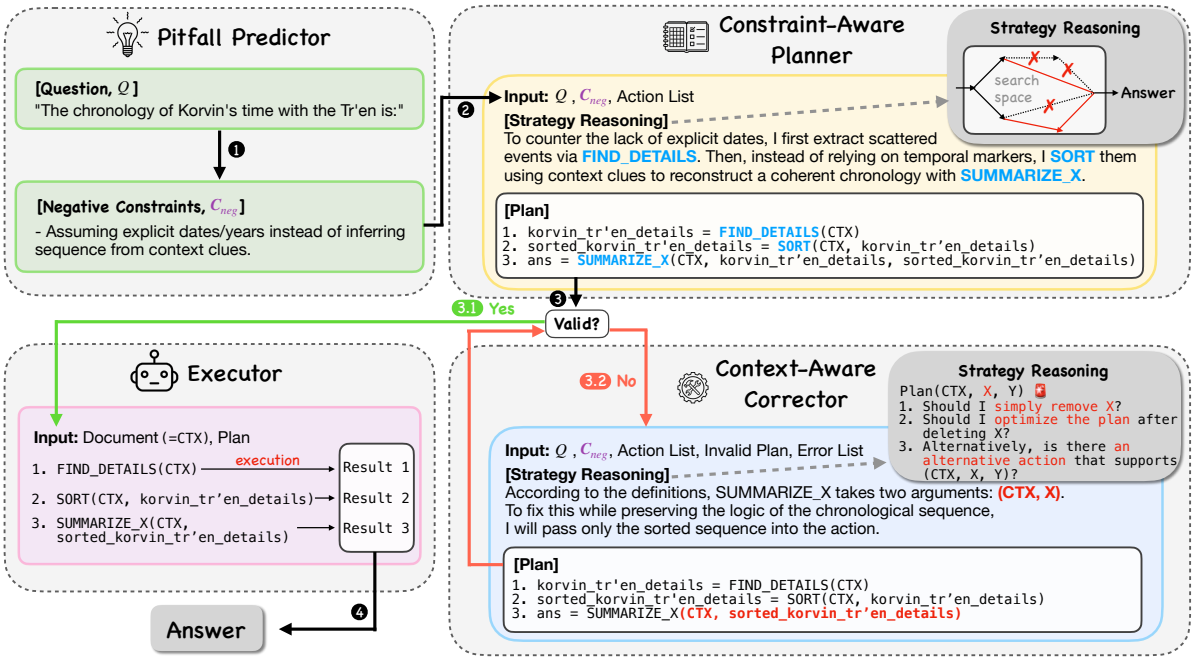


Figure 2: Overview of PPA-Plan, a proactive planning framework designed to generate reliable plans and execute them for long-context reasoning. The figure illustrates the full planning process through a concrete example. (1) If the document is not expected to contain explicit temporal markers based on the query, $\mathcal{M}_{pred}$ generates negative constraints to suppress the assumption of concrete dates. (2) Guided by these constraints, $\mathcal{M}_{plan}$ performs strategy reasoning to reconstruct the chronology from scattered events based on context clues, rather than attempting a futile search for nonexistent explicit dates, then $\mathcal{M}_{plan}$ generates a plan based on this strategy. (3) If the plan has an invalid format, such as incorrect arguments of SUMMARIZE_X, $\mathcal{M}_{corr}$ conducts strategy analysis to find the best way of fixing and optimizing the plan, such as by removing redundant arguments.

an executable plan. This approach prevents the generation of non-executable text, aligning the agent's logic with the environment's specific capabilities. The initial plan is generated as:

$$\mathcal{P}^{(0)} = \mathcal{M}_{plan}(q, \mathcal{A}, \mathcal{C}_{neg})$$

Once $\mathcal{P}^{(0)}$ is generated, a syntactic validity check is applied. If the plan is executable, it is forwarded to the execution module. Otherwise, it is passed to the corrector for refinement.

3.3 Context-Aware Correction

Simultaneously managing high-level logic and strict formatting often leads LLMs to prioritize semantic integrity over syntactic precision (Saparov and He, 2023; Zhao et al., 2025). To resolve these issues, we propose *Context-Aware Correction*, which decouples syntactic alignment from logical planning. The corrector $\mathcal{M}_{corr}$ receives q and $\mathcal{C}_{neg}$ alongside immediate feedback $\mathcal{F}^{(t-1)}$, defined as the invalid plan and associated error messages from the previous step $(t-1)$. By focusing on $\mathcal{F}^{(t-1)}$ over a cumulative history, we mitigate the risk of informational noise from stale iterations and prioritize diagnosing and repairing the current failure.

$\mathcal{M}_{corr}$ performs Strategy Reasoning by mapping the error feedback $\mathcal{F}^{(t-1)}$ to a repair operation to rectify the flawed plan $\mathcal{P}^{(t-1)}$ generated for q. $\mathcal{M}_{corr}$ accordingly determines the optimal resolution that satisfies syntactic requirements without compromising the original intent. Based on this derived strategy, $\mathcal{M}_{corr}$ then reconstructs the invalid plan $\mathcal{P}^{(t-1)}$ into a refined version to achieve syntactic alignment.

To implement this, we employ few-shot in-context learning with demonstrations that cover representative failure cases, such as unknown actions, undefined variables, and incorrect argument counts (see Table 6). By showcasing the causal link between specific error types and their optimal resolutions, these few-shot demonstrations facilitate the strategy reasoning required to transform raw error messages into precise, executable plan corrections.

We formalize the iterative correction process as:

$$\mathcal{P}^{(t)} = \mathcal{M}_{corr}(q, \mathcal{A}, \mathcal{C}_{neg}, \mathcal{F}^{(t-1)}),\ 1 \leq t \leq B$$

This process repeats until a plan satisfies the syntax requirements for execution or reaches the budget on the number of corrections, B.

3.4 Plan Execution

Once the final executable plan is obtained, it is passed to the plan execution module $\mathcal{M}_{exec}$. The executor operates by sequentially parsing each step of the plan. For each step, it constructs a specific execution prompt by integrating the defined action, arguments, and the long document $\mathcal{D}$. The resulting output from this execution is stored in a designated variable. Crucially, for steps requiring inputs from preceding actions, the executor resolves data dependencies by substituting variable placeholders with their corresponding stored results. This iterative process converts the logical plan into a set of concrete evidence $\mathcal{E}$, which constitutes the final system response.

4 Experiments

4.1 Experimental Setup

Model Selection. We evaluate performance different LLMs: GPT-4o-mini (OpenAI et al., 2024), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Qwen-2.5-14B-Instruct (Qwen et al., 2025). For Llama and Qwen models, we apply 8-bit quantization (Dettmers et al., 2022) to maximize inference efficiency while minimizing performance loss. This approach ensures optimal performance within limited VRAM resources. Specific hyperparameter settings and hardware specifications are described in the Appendix A.1.

Datasets and Task Formulation. To verify complex reasoning in long contexts, we use QuALITY (Pang et al., 2022), ConditionalQA (Sun et al., 2022), and LongReason (Ling et al., 2025) for multiple-choice tasks and Qasper (Dasigi et al., 2021) for free-form answer tasks. We simulate real-world scenarios by restricting models from accessing gold-standard information; instead, we convert all instances into generative tasks that require long-form answers. For Table 1, we evaluate methods on the intersection of samples for which all methods successfully generated outputs to ensure a fair qualitative comparison, while Table 2 includes the full test set to better evaluate overall robustness. For input length L exceeding the context length M, we truncate the input sequence S to preserve the document's extremities, as critical information is typically concentrated at the beginning and end (Liu et al., 2024): $S_{1:L} \rightarrow [S_{1:\lfloor M/2 \rfloor}; S_{L-\lfloor M/2 \rfloor-1:L}]$ (Bai et al., 2024b). Detailed dataset statistics are provided in the Appendix A.1.

Evaluation Metrics. We primarily calculate token-level recall (Lin, 2004) and Natural Language Inference (NLI) scores (Honovich et al., 2022; Chen and Eger, 2023) to evaluate the quality of generated answers from multiple perspectives. Recall measures how well the model includes core information from the gold standard without omissions.

NLI-based evaluation (Chen and Eger, 2023) assesses semantic equivalence through logical entailment, effectively capturing nuances in noisy reasoning sequences where surface-level string matching often fails (Balamurali and Cheng, 2025). Our NLI-based evaluation uses the entailment probability from DeBERTa-V3-Large (He et al., 2023). To identify the presence of correct information within long answers, we follow the granularity alignment strategy (Laban et al., 2022) by applying a sliding window and selecting the maximum entailment probability as the final score. For ease of interpretation and consistency with accuracy, NLI scores and Recall are scaled to a 0–100 range.

For ConditionalQA, where we focus on the binary (Yes/No) classification, we exclude recall evaluation because the highly restricted answer format makes token-level metrics inappropriate. For NLI-based evaluation, we format each question and its ground truth into a declarative statement (e.g., 'The answer to [X] is [Y]') to verify whether generated responses logically entail the correct answer.

For multiple-choice datasets, we also measure accuracy by using GPT-4o as a judge (Zheng et al., 2023) to map the generated responses to the available options, as it shows a high correlation with human judgment and identifies semantic equivalence more accurately than simple string matching. Table 10 and Appendix A.1 provide implementation details for these metrics. For Qasper, accuracy evaluation is excluded as the dataset primarily consists of free-form responses rather than multiple-choice questions.

Methods. We select Generative Question Answering (GQA), zero-shot Chain-of-Thought

	Method	QuALITY			Cond.QA		LongReason			Qasper		Overall		
		Acc	Rec	NLI	Acc	NLI	Acc	Rec	NLI	Rec	NLI	Acc	Rec	NLI
GPT-4o mini	GQA	71.6	49.0	23.8	80.7	27.2	61.8	66.9	50.6	65.8	37.5	71.4	60.6	34.8
	CoT	71.1	50.9	26.2	85.1	46.5	67.4	**68.1**	49.0	64.9	46.2	**74.5**	61.3	42.0
	PS	67.4	46.6	31.2	81.1	42.2	65.2	65.5	41.6	62.4	40.5	71.2	58.2	38.9
	PEARL	70.3	50.9	38.1	**85.3**	**53.2**	56.8	64.5	**72.1**	66.5	51.1	70.8	60.6	53.6
	PPA-Plan	**73.4**	**54.0**	**41.0**	81.3	50.3	**67.6**	66.6	69.9	**67.1**	**61.8**	74.1	**62.6**	**55.8**
Llama 3.1-8B	GQA	68.5	47.7	23.2	77.7	21.4	59.2	57.7	37.1	60.5	31.5	68.5	55.3	28.3
	CoT	68.1	46.0	16.5	**82.2**	39.6	55.8	55.0	29.5	61.3	36.6	68.7	54.1	30.6
	PS	66.7	44.7	16.9	80.8	25.7	48.3	42.0	33.1	58.6	36.2	65.3	48.4	28.0
	PEARL	68.5	51.4	48.5	80.1	**75.9**	55.8	56.9	76.2	66.0	74.8	68.1	58.1	68.9
	PPA-Plan	**70.8**	**53.9**	**50.5**	79.1	**75.9**	**67.9**	**62.3**	**77.9**	**67.3**	**75.6**	**72.6**	**61.2**	**70.0**
Qwen 2.5-14B	GQA	70.8	44.6	21.0	78.9	17.8	64.1	63.2	40.2	62.5	34.9	71.3	56.8	28.5
	CoT	68.3	45.3	22.1	83.1	36.7	67.0	64.9	43.2	63.4	36.6	72.8	57.9	34.7
	PS	69.5	42.6	26.5	81.5	33.9	61.1	60.5	30.9	62.8	39.0	70.7	55.3	32.6
	PEARL	72.7	48.4	37.7	81.4	**56.1**	60.1	61.2	66.6	63.6	46.1	71.4	57.7	51.6
	PPA-Plan	**75.1**	**49.8**	**39.6**	**83.9**	54.8	**72.3**	**66.3**	**68.9**	**67.1**	**56.4**	**77.1**	**61.1**	**54.9**

Table 1: Performance comparison on various datasets. **Bold** indicates the best performance among all compared methods (GQA, CoT, PS, PEARL, and **PPA-Plan**) for each LLM and dataset combination. Cond.QA, GQA, and PS denote ConditionalQA, Generative Question Answering, and Plan-and-Solve, respectively.

(CoT) (Wei et al., 2022), Plan-and-Solve (Wang et al., 2023), and PEARL (Sun et al., 2024) as baselines. For CoT and Plan-and-Solve, we evaluate the full reasoning process, relying on NLI-based metrics to mitigate potential recall bias. To ensure a consistent comparison, our method PPA-Plan adopts the same action space as PEARL. For both PEARL and PPA-Plan, we set the plan correction budget $B = 7$ to balance performance and inference efficiency. We also set the maximum number of negative constraints to $k = 3$. Specific prompts are provided in Table 7, Table 8, and Table 9. For both PEARL and the Executor of PPA-Plan, we adopt the original prompt configurations as specified in Sun et al. (2024).

4.2 Results

Table 1 shows that PPA-Plan consistently achieves competitive performance across models. It yields notable gain over PEARL on Llama-3.1-8B-Instruct (+4.5% acc, +3.1% recall) and achieves the best accuracy on Qwen-2.5-14B-Instruct (77.1%). The improvements in the NLI metric, which evaluates the validity of reasoning, are even more significant than those in standard accuracy. Compared to CoT, PPA-Plan increases overall NLI scores by 13.8 points for GPT-4o-mini and 20.2 points for Qwen-2.5-14B-Instruct. The results on Llama are particularly remarkable; our method raises the NLI score from 30.6 to 70.0, representing a more than two-fold increase. This substantial performance gap demonstrates that PPA-Plan identifies logical causalities via constraints rather than merely matching keywords or generating superficially plausible yet logically flawed sentences.

The corrective impact of this framework is most pronounced in open-source models with relatively limited reasoning capabilities, such as Llama and Qwen. Notably, Qwen achieved the highest accuracy of 77.1% across all experimental groups, showing that our approach effectively unlocks the model's latent reasoning potential. These results suggest that the proposed method is a robust solution for maximizing the latent reasoning capacity within small to medium-sized models.

On the ConditionalQA dataset, PPA-Plan occasionally falls short of the highest scores in specific metrics. This gap likely stems from the multi-faceted analysis of our planning process, which increases information richness to avoid bias. The NLI model often perceives this multi-faceted content as noise or information dilution, resulting in underestimated quantitative scores. Appendix A.2 provides a detailed analysis and quantitative proof of this phenomenon.

Model	$\mathcal{M}_{pred}$	$\mathcal{M}_{plan}$	$\mathcal{M}_{corr}$	LongReason Acc	LongReason Rec	LongReason NLI
GPT-4o mini	✓	✓	✓	**60.9**	**61.9**	**65.6**
	✓	✓	✗	47.5	48.8	51.4
	✓	Van.	✗	53.1	54.8	60.9
	✗	Van.	✗	37.7	42.7	48.9
Qwen 2.5-14B	✓	✓	✓	**59.8**	**57.0**	**60.4**
	✓	✓	✗	50.6	47.2	50.5
	✓	Van.	✗	48.5	43.7	53.0
	✗	Van.	✗	40.9	42.2	47.5

Table 2: Ablation study on the LongReason dataset using GPT-4o-mini and Qwen-2.5-14B-Instruct. We evaluate the contribution of each component. Van. denotes Vanilla, representing a standard planner without the strategy reasoning process.

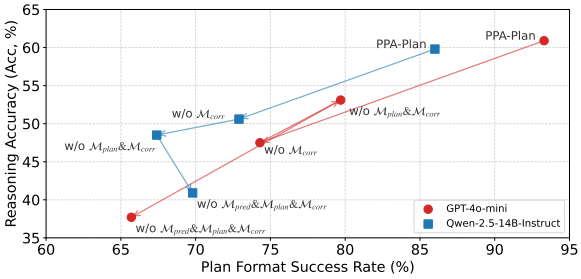


Figure 3: Impact of PPA-Plan components on plan executability and reasoning accuracy.

4.3 Ablation Study

This section analyzes how each component (the Pitfall Predictor $\mathcal{M}_{pred}$, the Constraint-Aware Planner $\mathcal{M}_{plan}$, and the Context-Aware Corrector $\mathcal{M}_{corr}$) contributes to the overall performance of PPA-Plan. Table 2 shows that for both GPT-4o-mini and Qwen-2.5-14B-Instruct, the full PPA-Plan configuration yields the best results across all metrics. We adopt the original prompt configurations for the Vanilla Planner, as specified in Sun et al. (2024).

Removing $\mathcal{M}_{pred}$ under the vanilla planner setup also results in a performance decline. For GPT-4o-mini, accuracy drops by 15.4% (from 53.1% to 37.7%) and the NLI score by 12 (from 60.9 to 48.9). A similar trend is observed in Qwen-2.5-14B-Instruct, where accuracy and NLI score decrease by 7.6% (from 48.5% to 40.9%) and by 5.5 (from 53.0 to 47.5). These results suggest that a planner struggles to identify logical pitfalls on its own. Thus, preemptive pitfall identification provides a critical guide for the planning process.

Removing $\mathcal{M}_{corr}$ also leads to a substantial performance drop, demonstrating its role as a safety net for plan executability. For GPT-4o-mini, accuracy decreases by 13.4% and the NLI score by 14.2; similarly, Qwen-2.5-14B-Instruct exhibits declines of 9.2% and 9.9. Interestingly, GPT-4o-mini performs worse when removing only $\mathcal{M}_{corr}$ (47.5%) than when removing the proposed $\mathcal{M}_{plan}$ and $\mathcal{M}_{corr}$ (53.1%). As shown in Figure 3, this occurs because sophisticated plans from $\mathcal{M}_{plan}$ often lead to format violations (Saparov and He, 2023; Zhao et al., 2025). Without $\mathcal{M}_{corr}$, the success rate of valid formats falls to 74.3%, lower than the baseline's 79.7%, whereas the full framework reaches 93.3%. This underscores that $\mathcal{M}_{\text{corr}}$ is essential for maintaining the validity of complex plans, creating an organic synergy between the proposed modules.

Conversely, Qwen shows an increased success rate when removing only $\mathcal{M}_{corr}$ (72.9%) compared to the baseline, where the proposed $\mathcal{M}_{plan}$ and $\mathcal{M}_{corr}$ are absent (67.4%). This trend indicates that the plan generated by Qwen's $\mathcal{M}_{plan}$ has relatively low complexity, which a baseline corrector can handle to some extent. These findings confirm that $\mathcal{M}_{corr}$ becomes increasingly vital for high-performance models; because these models generate more sophisticated plans, they require a specialized correction module to ensure executability without violating the required format.

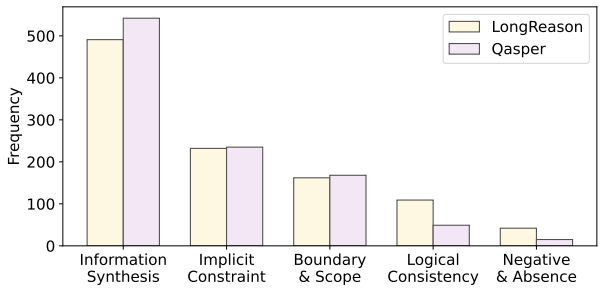
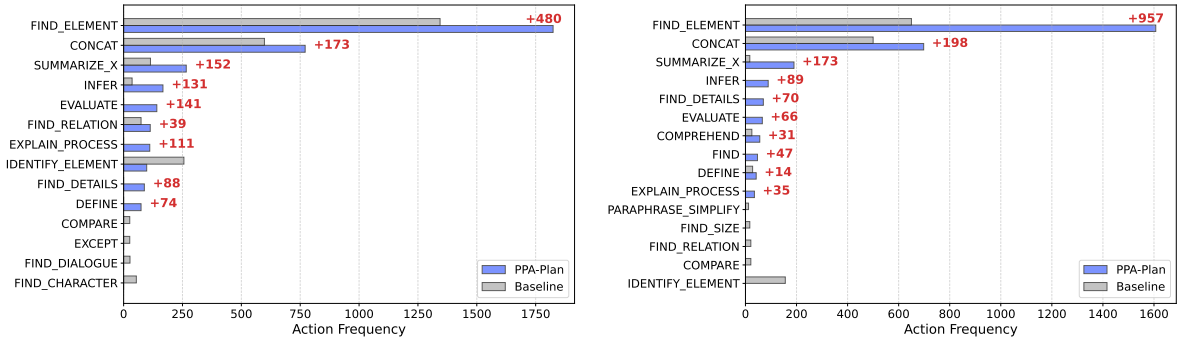


Figure 4: Distribution of negative constraint types generated by the Pitfall Predictor on LongReason and Qasper.

4.4 Analysis

Distribution of Negative Constraints. We analyzed the characteristics of negative constraints by classifying 200 sampled instances from LongReason and Qasper using GPT-4o across five categories. As shown in Figure 4, information synthesis is the most frequent type, followed by implicit constraints and boundary & scope. This distribution confirms that our framework identifies pitfalls requiring global information integration and logical depth beyond surface-level text. Full definitions and statistics are provided in Appendix A.3.



(a) LongReason

(b) Qasper

Figure 5: Strategic shift in action distributions induced by negative constraints and strategy reasoning. The baseline refers to a vanilla plan-and-execute setup that uses the Vanilla Planner without negative constraints and strategy reasoning, in contrast to PPA-Plan, which incorporates both.

Impact on Planning Behavior. This section examines how negative constraints transform the planner's reasoning. We compare PPA-Plan with a baseline that uses the Vanilla Planner without negative constraints and strategy reasoning using Qwen-2.5-14B-Instruct on LongReason and Qasper, focusing on the top 10 actions. Given the consistent patterns across both datasets, we use LongReason as a representative case.

Results show that applying negative constraints induces substantial changes in the model's reasoning process. Most notably, this approach improves plan complexity and sophistication. The average plan step count increases by 1.26 from 4.47 to 5.73 in LongReason and by 1.45 from 2.98 to 4.43 in Qasper compared to the baseline. This increase indicates that the model establishes more detailed and sophisticated plans to preemptively block potential errors by using negative constraints and strategy reasoning.

Second, action patterns shift from simple extraction toward logical reasoning. Figure 5 shows that high-level reasoning actions, such as INFER and SUMMARIZE_X, increase significantly. This trend suggests that constraints force the model to logically connect and summarize gathered information instead of simply listing it.

Third, the model's self-verification mechanism becomes active. EVALUATE and EXPLAIN_PROCESS appear frequently in plans using constraints, even though they were absent from the top baseline actions. This shift shows that the model does more than verify whether its plan satisfies the given constraints; it also facilitates a more objective assessment of the context, thereby mitigating potential reasoning biases.

Fourth, information-collection behaviors have advanced significantly. The frequency of entity search actions relying on surface-level information, such as FIND_CHARACTER, FIND_DIALOGUE, COMPARE, and IDENTIFY_ELEMENT, decreases. In contrast, deep evidence collection actions such as FIND_ELEMENT and FIND_DETAILS increase, helping the model collect specific evidence, identify logical contradictions, and explore logical interdependencies among various elements. This contrast indicates that the model avoids surface-level keyword searches and instead collects multifaceted logical evidence to improve the precision of its reasoning.

5 Conclusion

To address the tendency of LLM-based planners to anchor on initial flawed outputs during reactive refinement, we propose PPA-Plan, a planning strategy that proactively prevents logical pitfalls in planning by introducing negative constraints prior to plan generation. Experimental results demonstrate that PPA-Plan consistently achieves higher accuracy and stronger long-context reasoning capabilities than existing baselines. Our analysis further reveals that, by incorporating negative constraints with multiple perspectives, PPA-Plan encourages more deliberate planning behaviors, shifting from surface-level keyword search toward analytical actions such as evaluation and reasoning. In conclusion, this study establishes a systematic framework that allows LLMs to recognize pitfalls and think strategically before execution. PPA-Plan presents a new paradigm for ensuring reliability and accuracy in agent environments requiring complex planning.

Limitations

While PPA-Plan achieves impressive results, we identify three primary limitations. First, because the Context-Aware Corrector assumes the model can generate a baseline level of coherent output, small-scale models struggle to generate a valid plan format, despite the correction process. Second, the framework operates on the premise that negative constraints accurately capture actual pitfalls. Identifying a false pitfall introduces noise that disrupts the planning process, which prevents the model from establishing an accurate plan. Third, our system inherits the efficiency issues found in PEARL; repeatedly processing long contexts alongside multiple plan steps results in slow inference speeds.

To strengthen system robustness, we plan to explore fine-tuning techniques for smaller models and add a module to verify the logical validity of generated negative constraints. We also aim to ensure practical utility by applying context caching to increase inference efficiency.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024a. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024b. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137.

Sai Shridhar Balamurali and Lu Cheng. 2025. Revisiting nli: Towards cost-effective and human-aligned metrics for evaluating llms in question answering. *Preprint*, arXiv:2511.07659.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Yanran Chen and Steffen Eger. 2023. MENLI: Robust evaluation metrics from natural language inference. *Transactions of the Association for Computational Linguistics*, 11:804–825.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, Online. Association for Computational Linguistics.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Preprint*, arXiv:2208.07339.

Lutfi Eren Erdogan, Hiroki Furuta, Sehoon Kim, Nicholas Lee, Suhong Moon, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*.

Or Honovich, Roee Aharoni, Jonathan Herzig, Hagai Taitelbaum, Doron Kukliansy, Vered Cohen, Thomas Scialom, Idan Szpektor, Avinatan Hassidim, and Yossi Matias. 2022. TRUE: Re-evaluating factual consistency evaluation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3905–3920, Seattle, United States. Association for Computational Linguistics.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *Preprint*, arXiv:2404.06654.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32779–32798.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *International Conference on Machine Learning*, pages 24370–24391. PMLR.

Philippe Laban, Tobias Schnabel, Paul N Bennett, and Marti A Hearst. 2022. Summac: Re-visiting nli-based models for inconsistency detection in summarization. *Transactions of the Association for Computational Linguistics*, 10:163–177.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. 2024. Loogle: Can long-context language models understand long contexts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16304–16333.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Zhan Ling, Kang Liu, Kai Yan, Yifan Yang, Weijian Lin, Ting-Han Fan, Lingfeng Shen, Zhengyin Du, and Jiecao Chen. 2025. Longreason: A synthetic long-context reasoning benchmark via context expansion. *arXiv preprint arXiv:2501.15089*.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel Bowman. 2022. QuALITY: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5336–5358, Seattle, United States. Association for Computational Linguistics.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252.

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.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, and 1 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Haitian Sun, William Cohen, and Ruslan Salakhutdinov. 2022. ConditionalQA: A complex reading comprehension dataset with conditional answers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3627–3637, Dublin, Ireland. Association for Computational Linguistics.

Simeng Sun, Yang Liu, Shuohang Wang, Dan Iter, Chenguang Zhu, and Mohit Iyyer. 2024. PEARL: Prompting large language models to plan and execute actions over long documents. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1:*

Long Papers), pages 469–486, St. Julian's, Malta. Association for Computational Linguistics.

Ruixiang Tang, Dehan Kong, Longtao Huang, and Hui Xue. 2023. Large language models can be lazy learners: Analyze shortcuts in in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4645–4657, Toronto, Canada. Association for Computational Linguistics.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

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.

Michael Woolridge. 2001. Introduction to multiagent systems.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Weixiang Zhao, Xingyu Sui, Jiahe Guo, Yulin Hu, Yang Deng, Yanyan Zhao, Xuda Zhi, Yongbo Huang, Hao He, Wanxiang Che, and 1 others. 2025. Trade-offs in large reasoning models: An empirical analysis of deliberative and adaptive reasoning over foundational capabilities. *arXiv preprint arXiv:2503.17979*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 46595–46623.

A Appendix

A.1 Experimental Setup Details

All local experiments were conducted on a single NVIDIA A6000 48GB GPU. For all models, including GPT-4o (LLM-as-a-Judge), we employed greedy decoding (temperature = 0, do_sample = False) to ensure reproducibility. For all generation tasks, we set the maximum number of output tokens to 512, whereas the pitfall predictor was limited to 256 tokens. In the NLI-based evaluation, we set the window size to 512 and stride to 256. Since all method is training-free and requires no parameter updates, we utilized the entire set of available samples for evaluation, comprising the original training, validation, and test splits, to ensure statistical robustness. Adopting the setup from PEARL (Sun et al., 2024), we utilized the human annotation scores to distinguish task difficulty. An average score ≥ 3 were classified into the long split, while the remainder were classified into the short split. The detailed composition of our evaluation dataset is in Table 3.

Dataset	Config.	Split	Samples
QuALITY	long	dev	330
	long	train	368
	short	dev	302
			1000
Cond.QA	binary	train	860
	binary	dev	140
			1000
LongReason	16k	-	500
	32k	-	500
			1000
Qasper	free-form	dev	300
	free-form	train	600
			900

Table 3: Overall composition and statistics of the evaluation datasets used for the main results. Cond.QA denotes ConditionalQA dataset, Config. denotes Configuration.

A.2 Intrinsic Reasoning Faithfulness

To verify the logical robustness and correctness of the answers generated under our constraint sets, we conducted a separate core conclusion extraction analysis. As shown in Figure 6, using 200 randomly sampled instances from the ConditionalQA dataset, we compared the original answer scores (S_{total}) with the scores of core conclusions (S_{core}) extracted via GPT-4o-mini. We categorized these instances into low-score ($S_{\text{total}} < 0.3$) and high-score ($S_{\text{total}} > 0.7$) groups to measure the transition rates between these score states. Experimental results show that `PPA-Plan` achieves a 56.73% transition rate from the initial low S_{total} group to the high S_{core} group. This rate is more than 15% higher than those of CoT (40.54%) and PEARL (41.98%). This result proves that the abundance of information in `PPA-Plan`, generated to avoid bias or perform multi-perspective analysis, acts as noise during NLI evaluation rather than indicating actual logical failures. While `PPA-Plan` maintains logical robustness and correctness through deep reasoning, NLI metrics underestimate its performance more severely than other methods. In contrast, CoT shows the lowest transition rate, suggesting its low scores likely stem from actual logical flaws rather than information richness.

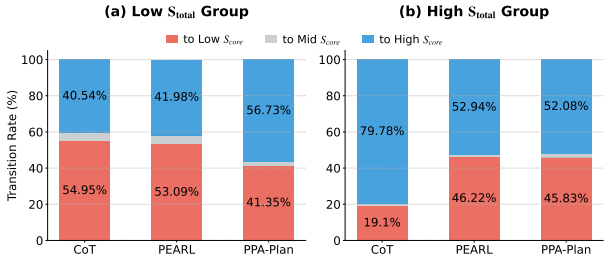


Figure 6: NLI transition analysis of S_{total} to S_{core}. (a) represents the recovery rate in the low-score group, while (b) shows the logical density and evidence retention in the high-score group. Note that `PPA-Plan` successfully bypasses potential pitfalls through multi-faceted reasoning.

Analysis of the high-score (S_{total}) group reveals that extracting core conclusions caused transitions to low or mid S_{core} at rates of 47.06% for PEARL and 47.91% for `PPA-Plan`. These figures represent a score drop approximately 27% larger than the 20.22% observed for CoT. This significant decrease indicates that the detailed explanations generated by `PPA-Plan` serve as essential logical grounding that supports the final answer. Notably, CoT exhibited the highest state-retention rates, 54.95% for low-to-low and 79.78% for high-to-high, confirming that NLI metrics underestimate CoT the least among all compared approaches.

A.3 Distribution of Negative Constraints.

This section analyzes the error types identified by the Pitfall Predictor and the characteristics of the resulting negative constraints. We randomly sampled 200 instances of negative constraints generated using Qwen-2.5-14B-Instruct from the LongReason and Qasper datasets. GPT-4o then served as an LLM judge to perform multi-label classification across five predefined categories (Figure 4).

Results show that information synthesis is the most frequent category in both datasets, with 491 labels in LongReason and 542 in Qasper. The total label count exceeds the number of samples because a single negative constraint can belong to multiple categories. This high frequency indicates that our constraints force the model to integrate scattered information throughout long contexts during reasoning. By requiring this synthesis, the framework prevents the model from drawing conclusions based solely on localized information within specific paragraphs.

Implicit constraints also account for a significant portion of the labels (232 and 235 occurrences, respectively). These constraints address essential preconditions for logical completeness not explicitly stated in the text. By identifying these hidden requirements, the negative constraints prevent models from merely reacting to surface-level text and instead force them to recognize underlying logical constraints for deeper reasoning.

Finally, the boundary & scope category (162 and 168 occurrences) optimizes the vast search space within the specific limits of the problem. This approach prevents premature generalizations or unfounded logical expansions, where models create unfounded causal links to jump to conclusions, allowing the model to maintain reasoning precision within the prescribed domain.

Negative Constraint Generation

```
You are an expert Exam Designer and Logic Analyst. Your goal is to look at the [Question] and
predict 1-3 critical pitfalls a planner might make related to risky logical assumptions.
Instead of stating obvious generalities, you must identify content-specific risks where a
simple keyword search or superficial reading would lead to a wrong answer.

Strict Constraints:
1. No Hallucination: Do not assume specific text structures (e.g., "split into two").
2. No Solutions: Identify the trap only. Do NOT provide actionable plans here.
3. No Repetition: Identify assumptions *unique* to this question, not just copying examples.

Return the result as a concise JSON list.
Format:
{"assumption_pitfalls": [
"<Pitfall 1: A brief explanation of the pitfall>",
"<Pitfall 2: (Optional)>",
"<Pitfall 3: (Optional)>"
]}

---
### Example 1 (Multiple Inferences Required)

[Question]
"Why did the author write the article?"

[Answer]
{"assumption_pitfalls": [
"Assuming the author's 'purpose' or 'reason' is stated explicitly as a single sentence.",
"Assuming the 'author' and the 'narrator' are the same entity and share the same motivations.",
"Assuming the purpose must be inferred only from the conclusion, and not from the article's
overall tone and main theme."
]}

---
### Example 2 (Specific Scope Constraint)

[Question]
"What is the "space cafard" that Si describes?"

[Answer]
{"assumption_pitfalls": [
"Assuming any general definition of 'space cafard' is correct, rather than focusing only on the
specific definition or description provided by Si."
]}

---
### Example 3 (Multiple Subjects / Scattered Info)

[Question]
"Arvid 6 and Tendal 13 can perform all of the following abilities EXCEPT:"

[Answer]
{"assumption_pitfalls": [
"Assuming the abilities for 'Arvid 6' and 'Tendal 13' are listed together in the same section.",
"Assuming a single list covers both characters, overlooking the possibility that their details
are distributed across different parts of the text."
]}

---
### Example 4 (Evaluation & Definition)

[Question]
"Of the following options, which seems to be Tremaine's biggest asset in his investigation?"

[Answer]
{"assumption_pitfalls": [
```

```
"Assuming the 'biggest' asset is explicitly labeled as such, rather than needing to first list
all assets and then infer their contribution.",
"Assuming 'asset' refers only to physical tools or skills, and overlooking abstract assets like
'intuition', 'connections', or 'reputation'."
]}

---
### Example 5 (Complex Prediction)

[Question]
"Out of the choices below, predict which future career Eddie would most likely pick given his
interests present in the article."

[Answer]
{"assumption_pitfalls": [
"Assuming the prediction relies solely on explicitly stated 'interests', overlooking implied
'skills' or 'aptitudes' mentioned in the text.",
"Assuming the decision is based only on positive factors, failing to account for tasks Eddie
explicitly 'dislikes' or 'avoids' which act as negative constraints.",
"Assuming the 'most likely' career is explicitly stated as a future goal, rather than requiring
a probabilistic prediction based on his comprehensive profile."
]}

---
### Example 6 (Counting / Manual Aggregation)

[Question]
"How many times has Critten been a Nilly?"

[Answer]
{"assumption_pitfalls": [
"Assuming the text provides a pre-calculated total count (e.g., 'three times'), rather than
requiring the reader to find and count individual mentions manually."
]}

---
### Your Task

[Question]
{question}

[Answer]
```

Table 4: Detailed prompt for the Pitfall Predictor of PPA-Plan.

Constraint-Aware Plan Generation

```
[Actions]
{action_list}
* Note: The output of each action can be the input to other actions.

[Instructions]
Suppose you are given a question about an article, as well as a list of potential actions (shown
above) that you can execute to solve the question. You can imagine the actions as functions in
a program, where you have input arguments, as well as output. The output of an action can be
fed as input to another action.
Please present a sequence of actions that you would use to answer the question after you read
the article. The sequence of actions should be specific and cover all the details about the
question. Please prioritize using the actions presented in the list above. If you need to add
new actions, please follow the format below. Please assign the output of each action with a
distinct name, which can be passed into other actions as argument.
Think twice before you provide your answer. Make sure your answer is valid, clear, and easy
to understand. Keep the answer simple and remove any unnecessary steps. Do not use list
comprehension or dictionary comprehension. Keep each action minimally simple. If a question is
unanswerable (e.g., requires options), collect as much information as possible from the input
such that it will be answerable when provided with options.
```

Since you will be provided with specific pitfalls that must be avoided, please start by performing a brief [Strategy Reasoning]. In this section, explicitly think about how to modify your plan or add specific actions to satisfy these constraints. Once you have established this strategy, proceed to generate the final sequence of actions in the [Answer] section.

Your answer should follow the format:
```
[Strategy Reasoning]
(Briefly explain how you will address the provided pitfalls here)

[Answer]
New actions:
- new_action_1(arguments) : [one-sentence general explanation] or "-None" if there no need to add new actions
- new_action_2(arguments) : [one-sentence general explanation] or "-None" if there no need to add new actions

1. output_1 = action_1(here goes arguments) : [one-sentence explanation]
2. output_2 = action_2(here goes arguments) : [one-sentence explanation]
...
```

The following are a few examples:

Question: "Why is Si retirement so significant to the Space Exploration Team?"
Input Pitfalls:
- Assuming the significance is stated in a single sentence explicitly linking retirement to the team.
- Ignoring the separate chain of events: the reason for retirement and its subsequent consequences.

[Strategy Reasoning]
The pitfalls warn against looking for a simple, direct link. To address this, I need to split the search: first find the 'cause' of retirement, then find the 'impact' of retirement separately. Then, I must explicitly connect both findings to the 'Space Exploration Team' to synthesize the full answer.

[Answer]
New actions:
- None

1. retire_reason = FIND_ELEMENT(CTX, "cause", "Si retirement") : Find and summarize the cause or reason of Si retirement from the input article
2. retire_outcome = FIND_IMPACTS(CTX, "Si retirement") : Find and summarize the impact or outcome or consequences of Si retirement from the input article
3. connect_reason = FIND_RELATION(CTX, retire_reason, "Space Exploration Team") : Find and summarize how the reason of Si retirement is related to the Space Exploration Team
4. connect_outcome = FIND_RELATION(CTX, retire_outcome, "Space Exploration Team") : Find and summarize how the outcome of Si retirement is related to the Space Exploration Team
5. ans = CONCAT(connect_reason, connect_outcome) : Combine the previous two steps to form the final answer

---
Question: "What is the "space cafard" that Si describes?"
Input Pitfalls:
- Assuming any general definition of 'space cafard' is correct.
- Failing to restrict the search to only Si's specific description provided in the text.

[Strategy Reasoning]
The constraint emphasizes avoiding general definitions. Therefore, I must restrict the 'FIND_ELEMENT' action to look specifically for "Si's description" of the term, ensuring the source of the definition is strictly from the character Si in the text.

[Answer]
New actions:
- None

1. space_cafard = FIND_ELEMENT(CTX, "Si's description", "space cafard") : Find and summarize all relevant information about the "space cafard" strictly as described by Si
2. space_cafard_cmprh = COMPREHEND(CTX, space_cafard) : Provide a comprehension about the "space cafard" based on the findings
3. ans = CONCAT(space_cafard, space_cafard_cmprh) : Combine to form the final answer

---
Question: "How many times has Critten been a Nilly?"
Input Pitfalls:
- Assuming the total count (e.g., '3 times') is explicitly stated in the text.
- Assuming the plan can just 'search' for a number.

[Strategy Reasoning]
The pitfall indicates that a simple search for a number will fail because the total count isn't explicit. I must first use 'FIND_ALL_ISSUES' to locate every individual instance where Critten was a Nilly, and then use 'COUNT_X' to manually count these instances.

[Answer]
New actions:
- FIND_ALL_ISSUES(CTX, X) : Find and summarize all the events where X occurs in the input article
- COUNT_X(CTX, X) : Count the number of times that X occurs in the input article

1. all_nilly = FIND_ALL_ISSUES(CTX, "Critten been a Nilly") : Find and summarize all individual events/mentions where Critten has been a Nilly
2. num_nilly = COUNT_X(CTX, all_nilly) : Count the number of times that Critten has been a Nilly given the collected events above

---
Question: "Out of the choices below, predict which future career Eddie would most likely pick given his interests present in the article."
Input Pitfalls:
- Assuming only explicitly stated 'interests' matter for the prediction.
- Assuming the prediction should be based only on positive factors, ignoring things he dislikes.

[Strategy Reasoning]
The constraints highlight that relying solely on "interests" is insufficient. I must modify the plan to actively search for "skills/aptitudes" (implied interests) and "dislikes/avoids" (negative constraints). These additional factors must be concatenated into the profile before making a prediction.

[Answer]
New actions:
- PREDICT_CAREER(CTX, X, Y) : Predict the future career given a person X's future career according to their interests or goals Y

1. eddie = IDENTIFY_ELEMENT(CTX, "Eddie") : Identify who Eddie is in the input article
2. eddie_interests = FIND_ELEMENT(CTX, "interests", eddie) : Find and summarize all the interests of Eddie
3. eddie_skills = FIND_ELEMENT(CTX, "skills and aptitudes", eddie) : Find demonstrated skills or aptitudes, as required to avoid the pitfall of missing implied traits
4. eddie_dislikes = FIND_ELEMENT(CTX, "dislikes and avoids", eddie) : Find tasks Eddie dislikes, as required to filter out unlikely careers
5. eddie_goals = FIND_INTENT(CTX, eddie) : Find and summarize the intent/purpose/goal of Eddie
6. eddie_profile = CONCAT(eddie_interests, eddie_skills, eddie_dislikes, eddie_goals) : Combine interests, skills, dislikes, and goals to build a complete profile
7. ans = PREDICT_CAREER(CTX, "Eddie", eddie_profile) : Predict the future career based on the comprehensive profile

---
Question: "Which word doesn't describe the security guard?"
Input Pitfalls:
- Assuming the plan should search for words that *do not* describe the guard directly.
- Failing to understand this is a 'NOT' (exclusion) question requiring a list of valid descriptions first.

```
[Strategy Reasoning]
The pitfall warns against searching for the negative directly. The correct strategy is to first
find all words that *DO* describe the security guard in the text. Then, the final answer (likely
comparing with options later) will be derived from knowing what *is* true.

[Answer]
New actions:
- None

1. security_guard = FIND_CHARACTER(CTX, "security guard") : Find and summarize the character
traits of the security guard
2. guard_descriptions = FIND(CTX, "descriptive words", "security guard") : Find the words that
ARE used to describe the security guard in the text
3. ans = CONCAT(security_guard, guard_descriptions) : Combine the traits and descriptions to
form a basis for exclusion


---
Question: "Of the following options, which seems to be Tremaine's biggest asset in his
investigation?"
Input Pitfalls:
- Assuming 'asset' refers only to physical tools.
- Assuming the 'biggest' asset is explicitly labeled as such.

[Strategy Reasoning]
To avoid the pitfall of focusing only on physical tools, I must explicitly instruct the
'FIND_ELEMENT' action to look for "assets including abstract ones (intuition, connections)".
Also, since the "biggest" isn't labeled, I need to 'SORT' the found assets based on their impact
to determine the ranking.

[Answer]
New actions:
- SORT(CTX, X): Sort the elements in X in ascending order with concise reasons, based on the
input article

1. tremaine = IDENTIFY_ELEMENT(CTX, "Tremaine") : Identify who Tremaine is in the input article
2. tremaine_assets = FIND_ELEMENT(CTX, "assets (physical and abstract)", tremaine) : Find all
assets, explicitly including abstract ones like intuition or connections
3. ranked_assets = SORT(CTX, tremaine_assets) : Sort the assets in ascending order of
importance/impact based on the text


[Question]
Now you are given a question about an article:
{question}
You MUST avoid these core pitfalls identified for this question:
{assumption_pitfall}

Please provide a plan (sequence of actions) that can arrive to the answer after reading the
article. Before generating the final plan, please briefly analyze how to address these pitfalls
in a [Strategy Reasoning] section. Then, provide the final sequence in the [Answer] section. As
the corresponding options are not provided for the question, when the question is not answerable
without the options, simply collect as much information as possible from the input such that it
will be answerable with the options. Make sure the plan you generate is valid and faithful to
the question.

[Strategy Reasoning]
```

Table 5: Detailed prompt for the Planner of PPA-Plan.

Context-Aware Correction

```
[Actions]
{action_list}
* Note: The output of each action can be the input to other actions.
```

[Instructions]
Suppose you are given a question about an article, as well as a list of potential actions (shown above) that you can execute to solve the question. You can imagine the actions as functions in a program, where you have input arguments, as well as output. The output of an action can be fed as input to another action.
Please present a sequence of actions that you would use to answer the question after you read the article. The sequence of actions should be specific and cover all the details about the question. Please prioritize using the actions presented in the list above. If you need to add new actions, please follow the format below. Please assign the output of each action with a distinct name, which can be passed into other actions as argument. Think twice before you provide your answer. Make sure your answer is valid, clear, and easy to understand. Keep the answer simple and remove any unnecessary steps. Do not use list comprehension or dictionary comprehension. Keep each action minimally simple. If a question is unanswerable (e.g., requires options), collect as much information as possible from the input such that it will be answerable when provided with options.
Since you will be provided with an invalid plan and parser error messages, please start by performing a brief [Strategy Reasoning]. In this section, identify the cause of the error and plan how to fix the syntax while preserving the original logic. Once you have established this repair plan, proceed to generate the corrected sequence of actions in the [Answer] section.

Your answer should follow the format:
```
[Strategy Reasoning]
(Briefly explain how you will address the provided pitfalls here)

[Answer]
New actions:
- new_action_1(arguments) : [one-sentence general explanation] or "-None" if there no need to add new actions
- new_action_2(arguments) : [one-sentence general explanation] or "-None" if there no need to add new actions

1. output_1 = action_1(here goes arguments) : [one-sentence explanation]
2. output_2 = action_2(here goes arguments) : [one-sentence explanation]
...
```

The following are examples of how to correct an invalid plan based on error messages:

Example 1 (Error: Unknown Action)
Question: "What is the primary diet of the spectacled bear?"

Invalid Plan:
1. bear_info = FIND_ELEMENT(CTX, "diet", "spectacled bear") : Find diet info
2. ans = COMPREHEND(CTX, bear_info) : Understand the info

Error Message:
"Error parsing action COMPREHEND. Unknown action. Please define it in the 'New actions' section if needed, or choose from the existing action list."

Input Pitfalls:
- Assuming the diet consists of only one type of food.

[Strategy Reasoning]
The parser reports that ‘COMPREHEND‘ is an unknown action. Checking the valid action list, the correct action for summarizing information is ‘SUMMARIZE‘. I will replace ‘COMPREHEND‘ with ‘SUMMARIZE‘ while keeping the arguments the same to preserve the flow.

[Answer]
New actions:
- None

1. bear_info = FIND_ELEMENT(CTX, "diet", "spectacled bear") : Find diet info
2. ans = SUMMARIZE(CTX, bear_info) : Summarize the findings to form the answer

---

```
### Example 2 (Error: Undefined Variable)
Question: "How did the protagonist escape the room?"

Invalid Plan:
1. room_desc = FIND_ELEMENT(CTX, "escape method", "protagonist") : Find escape details
2. ans = GENERATE_ANSWER(CTX, room_info) : Generate the final answer

Error Messages:
"Error parsing action GENERATE_ANSWER. Argument room_info is not defined."

Input Pitfalls:
- Assuming the escape happened in a single step.

[Strategy Reasoning]
The error states that ‘room_info‘ is undefined. Looking at the previous step (step 1), the
output variable was named ‘room_desc‘, not ‘room_info‘. I must correct the argument in step 2
to match the variable name defined in step 1 (‘room_desc‘).

[Answer]
New actions:
- None

1. room_desc = FIND_ELEMENT(CTX, "escape method", "protagonist") : Find escape details
2. ans = GENERATE_ANSWER(CTX, room_desc) : Generate the final answer


---
### Example 3 (Error: Incorrect Argument Count)
Question: "List all the awards won by the author."

Invalid Plan:
1. awards = FIND_ALL_ISSUES("awards", "author") : Find all awards
2. ans = LIST_ITEMS(CTX, awards) : List them

Error Message:
"Error parsing action FIND_ALL_ISSUES. Number of arguments is incorrect"

Input Pitfalls:
- Assuming the awards are listed in a distinct 'awards' section.

[Strategy Reasoning]
The action ‘FIND_ALL_ISSUES‘ caused an argument count error. Standard actions usually require
‘CTX‘ as the first argument. I will add ‘CTX‘ to the arguments of ‘FIND_ALL_ISSUES‘ to satisfy
the function definition.

[Answer]
New actions:
- None

1. awards = FIND_ALL_ISSUES(CTX, "awards", "author") : Find all awards
2. ans = LIST_ITEMS(CTX, awards) : List them


---
### Example 4 (Error: Missing Action Definition)
Question: "Based on the historical data provided, predict the stock price for next month."

Invalid Plan:
1. history = FIND_DATA(CTX, "stock price history", "last 5 years") : Retrieve data
2. prediction = PREDICT_TREND(CTX, history) : Predict future price
3. ans = GENERATE_ANSWER(CTX, prediction) : Formulate answer

Error Message:
"Error parsing action PREDICT_TREND. Unknown action. Please define it in the 'New actions'
section if needed, or choose from the existing action list."

Input Pitfalls:
"Assuming a linear trend without considering volatility mentioned in the text."
```

```
[Strategy Reasoning]
The parser indicates PREDICT_TREND is unknown. This action is essential for the prediction
task and has no equivalent in the standard action list. Therefore, instead of replacing it, I
will retain PREDICT_TREND and explicitly define it in the 'New actions' section to fix the error.

[Answer]
New actions:
- PREDICT_TREND(CTX, data) : Analyze the provided data to forecast future trends or values.

1. history = FIND_DATA(CTX, "stock price history", "last 5 years") : Retrieve data
2. prediction = PREDICT_TREND(CTX, history) : Predict future price based on the retrieved
history
3. ans = GENERATE_ANSWER(CTX, prediction) : Formulate the final answer


[Question]
Given the following question,
Question: {question}
you just came up with the following sequence of actions as well as potential new actions:
{invalid_plan}

However, the above answer is invalid according to a parser, which returned an error message:
{error_message}

You MUST avoid these core pitfalls identified for this question:
{assumption_pitfall}

Can you edit the plan (sequence of actions) to make it valid? Please focus strictly on correcting
the syntax errors reported by the parser, such as missing arguments, missing descriptions, or
incorrect formatting. Do not attempt to generate a new strategy or change the sequence of
actions unless it is absolutely required to fix the syntax error (e.g., adding a missing step
to define an undefined variable). Your goal is to simply repair the invalid plan so that it
becomes parseable while maintaining its original logic.

Before generating the final corrected plan, please use the [Strategy Reasoning] section to
briefly identify the specific syntax error from the error message and explain how you will
correct the format (e.g., adding missing arguments or fixing the action definition) to satisfy
the parser. Then, provide the final corrected sequence in the [Answer] section.

[Strategy Reasoning]
```

Table 6: Detailed prompt for the Corrector of PPA-Plan.

Generative Question Answering (GQA)

```
Article

{article}

End of Article


Question: {question}

Answer:
(Please provide a detailed explanation for answering the question above.)
```

Table 7: Detailed prompt for Generative Question Answering (GQA).

Chain-of-Thought (CoT)

```
Article:
{article}

Question: {question}

Please think step by step to find the answer based on the article.
Provide your reasoning process first, and then give the final answer.

Reasoning:
```

Table 8: Detailed prompt for Chain-of-Thought (CoT).

Plan-and-Solve

```
Article:
{article}

Question: {question}

Let's first understand the problem and devise a plan to solve it. Then, let's carry out the
plan and solve the problem step by step. Please respond in the following format:
Plan: [Your plan here]
Solution: [Your step-by-step execution here]

Answer:
```

Table 9: Detailed prompt for Plan-and-Solve.

LLM-as-a-Judge

```
Relevant information for answering the question:

{open_answer}

Question:{question}
{options}

Carefully compare all four options (A, B, C, and D) based on the relevant information. Select
the best possible answer by ensuring it is the most accurate choice given the information
provided. Write only the letter of your final answer without explanation. Answer (select from
A, B, C, D):
```

Table 10: Detailed prompt for LLM-as-a-Judge in multiple-choice questions.