POKE: PARTIAL ORDER KNOWLEDGE ENGINEERING FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

To improve the rigid and step-by-step prompt engineering for large language models (LLMs), we propose the Partial Order Knowledge Engineering (POKE), which is a flexible and self-evolving planning-based prompt framework to handle the multi-hop inference. In our analysis of the manual prompts, the hand-crafted presentation of auxiliary knowledge is the key to the high-quality responses. The conventional prompt generation processes would be similar to solve the planning problems. In the POKE framework, we train a self-evolving planner to explicitly organize the knowledge of the LLMs. To apply POKE, the learned planner constructs a partial-order plan over the knowledge modules, the ordering constraints, and the causal links, after which an executor propagates the resolved intermediate plan and generates the prompt. We evaluate the POKE method on three atomic-probe subsets of TwoHopFact, MuSiQue, and 2WikiMultiHopQA. Experiments on the shared TwoHopFact subset prove that the POKE improves accuracy over the state-of-the-art methods. We also demonstrate that the POKE framework is capable to align with the physical laws and the structural constraints, by successfully fulfilling the physics-based simulation script generation tasks.

1 INTRODUCTION

Large language models (LLMs) have achieved increasing capabilities in reasoning, instruction following, and task generalization as the scale of the models ever-growing (Touvron et al., 2023a;b). Prompting methods such as Chain-of-Thought (CoT) further improve multi-step reasoning by encouraging the models to explicitly generate intermediate steps (Wei et al., 2023). However, stronger models and longer reasoning traces do not guarantee that the knowledge required by a problem could be reliably composed during inference. In many cases, although the model may have useful knowledge relevant to a given question, it still fails to apply that knowledge to a coherent reasoning process (Yang et al., 2025b; Zhang et al., 2026). From this perspective, hallucination and low-quality generation may arise not only because of the knowledge absence, but also because of the under-activation of the knowledge existing in the model. This knowledge-to-apply gap does not consistently narrow down with model scaling (Press et al., 2023).

One possible origin of this knowledge-to-apply gap lies in the mismatch between improperly organized prompts and the dependency structure of complex reasoning. Conventional prompt techniques provide reasoning guidance for LLMs through a manually pre-defined trajectory. However, the pre-defined trajectory may be insufficient or even conflict with the LLMs' native reasoning steps. For example, suppose that there are two independent knowledge components a and b required to derive c, the true dependency structure is $a \rightarrow c$ and $b \rightarrow c$. In such a case, some particular sequence such as $a \rightarrow b \rightarrow c$ or $b \rightarrow a \rightarrow c$ would confuse the LLMs, even though these prompts seem to be sufficient to human experts. In our analysis, committing prematurely to one sequence could introduce unnecessary ordering relations, omit relevant knowledge components, or propagate an incorrect intermediate dependency. We argue that the conventional CoT methods are flawed, because those methods neglect the knowledge dependency native in LLMs.

To address the limitations of manually generated prompts, we propose the **Partial Order Knowledge Engineering (POKE)**, a planning-based framework that applies Partial Order Planning (POP) to LLM generation. Given a question, a learned planner constructs a partial-order plan $\Pi = \langle A, O, L \rangle$, where A specifies the required knowledge modules, O contains only the ordering con-

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

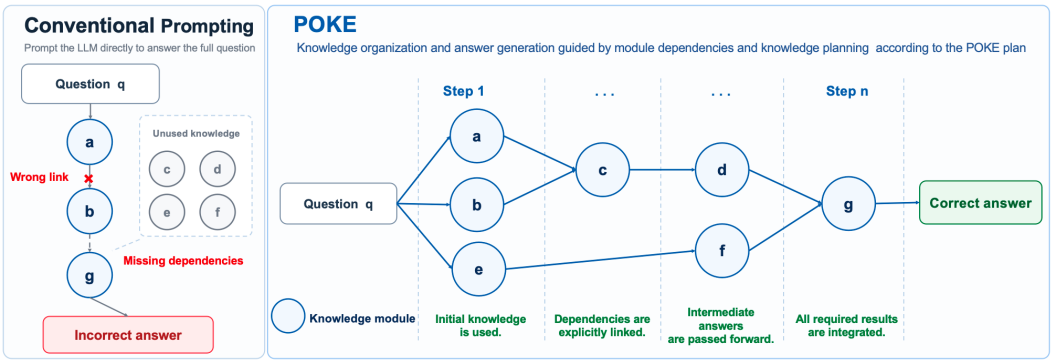


Figure 1: Illustration of POKE. Conventional inference implicitly constructs a single reasoning chain during generation and may miss required knowledge components or introduce incorrect dependencies. Instead, POKE organizes existing knowledge into a partial-order dependency structure, allowing multiple prerequisite modules to be resolved independently and progressively composed into a valid solution path.

straints necessary for execution, and L specifies the dependency links through which intermediate results are passed between modules. Independent modules can therefore remain unordered and be resolved separately, while dependent modules are executed only after their prerequisites have been satisfied. LLM then executes a valid trajectory of the plan and progressively propagates resolved intermediate outputs to downstream reasoning. As illustrated in Figure 1, POKE does not introduce external knowledge. Instead, it explicitly organizes knowledge, which is native to the LLMs, so that the required components are selected, connected, and composed according to their dependencies. In this sense, POKE acts as an external knowledge-organization agent during inference, aiming to improve reasoning quality without increasing the model size or relying on external knowledge. Through this structured organization, POKE helps the LLMs better activate, expose, connect, and utilize knowledge to conduct proper reasoning. We therefore argue that substantial improvements in reasoning could be achieved not only through model scaling, but also through more effective engineering of knowledge activation and composition during inference, shifting the focus from parameter expansion toward knowledge orchestration and the more efficient use of knowledge already encoded in the LLMs.

The main contributions of this work are two-fold:

- We identify and systematically study a knowledge-organization gap in LLM reasoning, i.e., successful retrieval of the constituent knowledge required by a problem does not necessarily lead to successful compositional reasoning.
- We propose the **Partial Order Knowledge Engineering (POKE)**, an inference-time framework that explicitly represents knowledge modules, ordering constraints, and causal dependency links, enabling LLMs to organize and utilize their existing knowledge according to a partial-order plan.

2 Related Work

2.1 Prompting and Structured Reasoning

Prompt engineering elicits task-relevant behavior from pretrained large language models without updating their parameters (Brown et al., 2020; Kojima et al., 2023). Zero-shot and few-shot prompts activate relevant knowledge through instructions or demonstrations (Lu et al., 2022). Instruction tuning and reinforcement learning from human feedback further improve instruction following through parameter optimization (Wei et al., 2022; Ouyang et al., 2022). Chain-of-Thought (CoT) prompting elicits intermediate reasoning steps before answer generation (Wei et al., 2023). Self-consistency, least-to-most prompting, and automatic CoT extend this process through path aggregation, staged decomposition, and demonstration generation (Wang et al., 2023c; Zhou et al., 2023; Zhang et al.,

2022). Tree of Thoughts (TOT) and Graph of Thoughts organize reasoning as tree or graph search (Yao et al., 2023; Besta et al., 2024), while SELF-DISCOVER and Program-Aided Language Models construct task-specific structures or executable programs (Zhou et al., 2024b; Gao et al., 2023). Self-evaluation and symbolic constraints further improve reasoning reliability (Xie et al., 2023; Xu et al., 2024). Recent work also extracts latent CoT paths through alternative decoding or performs reasoning in continuous hidden spaces (Wang & Zhou, 2024; Hao et al., 2025; Wu et al., 2026).

These methods improve elicitation, representation, and search for reasoning. However, they rarely specify prerequisite, causal, containment, and ordering dependencies among knowledge modules before execution. POKE addresses this gap through explicit partial-order knowledge organization.

2.2 Planning-Augmented Reasoning for Large Language Models

Planning-augmented approaches separate plan construction from answer or action generation. Plan-and-Solve decomposes problems into subproblems, while zero-shot planners translate high-level goals into intermediate actions (Wang et al., 2023a; Huang et al., 2022b). However, generated plans may violate action or state-transition constraints. PlanBench confirms persistent weaknesses in plan generation and reasoning about state changes (Valmeekam et al., 2023). Surveys characterize this direction as a shift toward planner-augmented, search-based, and structure-aware inference (Cao et al., 2025; Aghzal et al., 2025).

Feedback-driven methods further refine plans during interaction. AdaPlanner revises the generated plans according to environmental feedback (Sun et al., 2023), while DEPS ranks parallel subgoals according to their estimated achievability (Wang et al., 2024). Zhou et al. (2024a) combine tree search, value estimation, and self-reflection. World Knowledge Models guide global and local planning using knowledge synthesized from expert and sampled trajectories (Qiao et al., 2025). These methods primarily optimize sequential decisions in interactive environments.

POKE draws more directly from classical partial-order planning (POP), which retains only the ordering constraints required for goal satisfaction (Weld, 1994; Penberthy & Weld, 1992). Its least-commitment strategy naturally represents independent, branching, and converging subgoals. POKE applies this principle to inference-time knowledge composition by representing knowledge modules, ordering constraints, and causal links as $\Pi = \langle A, O, L \rangle$. The plan organizes knowledge already available to the executor and guides generation without parameter updates.

2.3 Knowledge Distillation and Relational Knowledge Transfer

Knowledge distillation transfers predictive behavior from a teacher model to a smaller student, commonly through output-distribution matching (Hinton et al., 2015). Its effectiveness also depends on data geometry and optimization behavior (Phuong & Lampert, 2021). Relational distillation extends output matching by preserving pairwise geometry, activation similarities, distributional relations, class distinctions, token correlations or multimodal dependencies (Park et al., 2019; Tung & Mori, 2019; Lv et al., 2024; Zhang et al., 2025; Gong et al., 2025; Cai et al., 2025).

Transformer-oriented methods transfer attention distributions and value relations, while correlation-based objectives preserve inter-class and intra-class structures (Wang et al., 2020; Huang et al., 2022a). Teacher-assistant methods further reduce the capacity gap between teacher and student models (Mirzadeh et al., 2019). Reasoning-oriented approaches such as SCOTT and KARD transfer self-consistent or knowledge-augmented rationales to smaller models (Wang et al., 2023b; Kang et al., 2023). Together, these studies support the view that knowledge is encoded partly through relationships among knowledge units (de Saussure et al., 1960).

Most relational distillation methods encode numerical similarity, correlation, or attention alignment into student parameters through training. POKE constructs dependencies among existing knowledge modules at inference time. It explicitly represents prerequisite, causal, containment, and ordering relations while leaving the executor parameters unchanged.

3 METHOD

3.1 POKE OVERVIEW

We propose the **Partial Order Knowledge Engineering (POKE)**, a plan-space reasoning framework to improve the LLM answer generation without updating the parameters of the base model. Given a question q, POKE constructs a partial-order plan over knowledge modules and executes them using an LLM. The plan specifies the prerequisites of each module and identifies which intermediate outputs satisfy those prerequisites. Execution follows a valid trajectory of the plan and passes outputs along the specified causal links.

3.2 KNOWLEDGE MODULES AND PLAN REPRESENTATION

Let I_q denote the conditions available from the input, G_q the answer requirements, and $\mathcal{M}_q$ the candidate module set. Each module is a planning action

$$m_i = \langle \mathrm{name}_i, \mathrm{type}_i, \mathrm{Pre}_i, \mathrm{Eff}_i^+, \mathrm{Eff}_i^-, \mathrm{prompt}_i \rangle, \tag{1}$$

where Pre_i specifies its prerequisites, Eff_i^+ and Eff_i^- specify expected effects, and prompt_i supplies the executor instruction. Positive effects establish conditions, while negative effects invalidate them within the planning representation. Modules describe individual reasoning operations, such as answering factual subquestions or combining intermediate outputs. Following partial-order planning (Weld, 1994; Penberthy & Weld, 1992), we represent a plan as

$$\Pi = \langle A, O, L \rangle, \qquad A \subseteq \mathcal{M}_q \cup \{m_{\mathrm{start}}, m_{\mathrm{finish}}\}. \tag{2}$$

Here, A selects the modules, O specifies their ordering constraints, and L records causal support. A causal link $m_i \xrightarrow{p} m_j$ requires $p \in \mathrm{Eff}_i^+ \cap \mathrm{Pre}_j$ and imposes $m_i \prec m_j$. The constraints must admit a valid trajectory, so $\mathrm{Lin}(A, O) \neq \emptyset$. Modules remain unordered unless dependencies or consistency constraints require an order.

3.3 PLAN CONSTRUCTION

POKE initializes a null partial-order plan with two special modules: m_{start}, which establishes the initial conditions I_q, and m_{finish}, whose preconditions are the answer goals G_q:

$$\mathrm{Pre}_{\mathrm{start}} = \emptyset, \quad \mathrm{Eff}^+_{\mathrm{start}} = I_q, \quad \mathrm{Pre}_{\mathrm{finish}} = G_q, \quad \mathrm{Eff}^+_{\mathrm{finish}} = \emptyset. \tag{3}$$

We initialize

$$\Pi_0 = \langle \{m_{\mathrm{start}}, m_{\mathrm{finish}}\}, \{m_{\mathrm{start}} \prec m_{\mathrm{finish}}\}, \emptyset \rangle . \tag{4}$$

The agenda contains the preconditions without an incoming supporting causal link:

$$\Gamma = \{\langle p, m \rangle \mid m \in A,\ p \in \mathrm{Pre}_m,\ p \text{ is unsupported}\} . \tag{5}$$

Initially, $\Gamma_0 = \{\langle g, m_{\mathrm{finish}} \rangle \mid g \in G_q\}$.

For an open condition $\langle p, m_{\mathrm{need}} \rangle$, select a supporter $m_{\mathrm{add}} \in A \cup \mathcal{M}_q$ with $p \in \mathrm{Eff}^+_{\mathrm{add}}$. The refinement updates

$$A' = A \cup \{m_{\mathrm{add}}\}, \quad L' = L \cup \{m_{\mathrm{add}} \xrightarrow{p} m_{\mathrm{need}}\}, \quad O' = O \cup \{m_{\mathrm{add}} \prec m_{\mathrm{need}}\}. \tag{6}$$

For a newly added module, we also impose $m_{\mathrm{start}} \prec m_{\mathrm{add}} \prec m_{\mathrm{finish}}$. The satisfied condition is removed from Γ, and unsupported prerequisites of the new module are added.

Consider a causal link $m_p \xrightarrow{p} m_c$, a module m_t threatens this link if $p \in \mathrm{Eff}_t^-$ and the current ordering constraints permit m_t to occur between m_p and m_c. POKE resolves the threat through demotion $m_t \prec m_p$ or promotion $m_c \prec m_t$, provided that the resulting ordering constraints remain consistent. Otherwise, the planner selects an alternative module or backtracks to an earlier refinement decision. A plan is complete when $\Gamma = \emptyset$, its ordering constraints admit a trajectory, and no unresolved threat remains.

3.4 PLAN-GUIDED EXECUTION

For a completed plan $\Pi^* = \langle A^*, O^*, L^* \rangle$, topological sorting yields an execution sequence $\tau \in \mathrm{Lin}(A^*, O^*)$. Each ordinary module receives the outputs specified by its incoming causal links:

$$\begin{aligned} \mathcal{C}_i &= \{(p, y_j) \mid (m_j \xrightarrow{p} m_i) \in L^*\}, \\ y_i &= f_\theta(q, \mathrm{prompt}_i, \mathcal{C}_i). \end{aligned} \tag{7}$$

For links from m_{start}, the supporting information is taken directly from I_q. At a join, $\mathcal{C}_i$ collects the required outputs from all supporting branches before execution.

After each module call, $\mathrm{Verify}(y_i, \mathrm{Eff}_i^+)$ checks the expected effects. If the check fails, the condition is reopened and an alternative supporter is selected. The finish module produces the final answer from its supporting outputs:

$$\hat{a} = f_\theta(q, \mathrm{prompt}_{\mathrm{finish}}, \mathcal{C}_{\mathrm{finish}}). \tag{8}$$

For physics-based simulation scripts, the plan additionally specifies object bindings, containment relations, and physical and numerical parameters, as detailed in Section 4.5.

4 EXPERIMENTS

We first investigate whether LLMs exhibit a knowledge-organization gap in multi-hop reasoning. Specifically, we examine cases in which a model correctly answers each constituent atomic question independently but fails to answer the complete compositional question. Such cases show that successful retrieval of individual facts does not necessarily lead to successful composition, motivating an evaluation of how the model connects, orders, and uses those facts during inference. We then evaluate whether POKE can narrow this gap by explicitly planning the dependencies and execution order among knowledge modules. Our results show that POKE improves compositional accuracy and, on a shared subset where all compared models correctly answer the required atomic probes, enables a smaller executor to outperform larger executors using conventional prompting methods (Brown et al., 2020; Zhang et al., 2022; Press et al., 2023). We further examine longer questions with chain and join dependencies and extend POKE to physics-based simulation script generation.

4.1 EXPERIMENTAL SETUP

Datasets. We use TwoHopFact as the primary diagnostic and evaluation dataset because each example explicitly specifies two atomic facts, $e_2 = r_1(e_1)$, $e_3 = r_2(e_2)$, where e_2 is the bridge entity connecting the two knowledge modules (Yang et al., 2025b). We also use 2-4 hop questions from MuSiQue to evaluate reasoning across different numbers of hops and chain and join dependency structures (Trivedi et al., 2022). We additionally evaluate POKE on questions from 2WikiMultiHopQA (Wiki) (Ho et al., 2020). Detailed dataset construction, subset definitions, and their roles in our experiments are provided in Appendix A.

Atomic Knowledge Probing. For each two-hop question, we query model M independently on the two constituent atomic questions. We define the model-specific subset

$$S_{11}^M = \{q : \mathrm{Correct}_M(q_1) = 1 \wedge \mathrm{Correct}_M(q_2) = 1\}. \tag{9}$$

Membership in S_{11}^M indicates that both atomic probes were answered correctly under the specified evaluation protocol. The subset allows us to examine compositional accuracy conditional on successful atomic probing. We compare inference methods on the same S_{11}^M subset for each executor. For cross-scale comparisons, we use the intersection of the subsets of the compared models: $S_{11}^{\mathrm{shared}} = \bigcap_{M \in \mathcal{E}} S_{11}^M$, where $\mathcal{E}$ denotes the set of executors included in the comparison. For 3-4 hop questions, we also require correct responses to all constituent probes, defining S_{111}^M and S_{1111}^M.

Models. Our primary experiments use models from the open-source Qwen3 dense family (Yang et al., 2025a). Qwen3 uses strong-to-weak distillation for its lightweight models, providing a useful setting for studying the relationship between model scale and knowledge organization. Models of different sizes may share substantial knowledge, while differing in both their overall knowledge coverage and their ability to compose that knowledge during inference. We therefore evaluate each

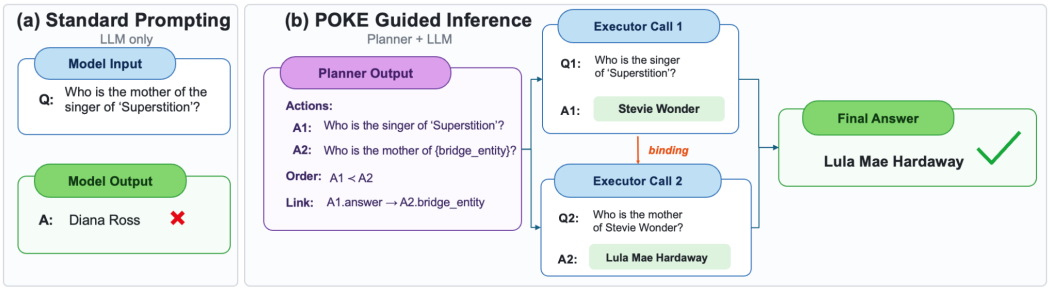


Figure 2: Schematic illustration of the knowledge-organization gap and POKE-guided inference. Under standard prompting, the model directly answers the composed question and must implicitly recover the required intermediate dependency. By contrast, POKE explicitly specifies the knowledge actions, their execution order, and the dependency link, and progressively binds the resolved intermediate result during execution.

model on its own atomic-probe subset and compare models of different scales on their shared subset. The shared-subset evaluation controls for question-level differences in atomic-probe success and allows us to examine whether explicit knowledge organization enables a smaller executor to match or outperform larger executors. To examine whether these gains extend beyond the primary Qwen3 models, we also evaluate DeepSeek-R1-0528-Qwen3-8B, DeepSeek-R1-Distill-Qwen-1.5B, DeepSeek-R1-Distill-Qwen-7B, and GLM-4-9B-0414 (Guo et al., 2025; GLM et al., 2024).

Baselines. We compare POKE with Direct prompting, zero-shot Chain-of-Thought (CoT) (Kojima et al., 2023), and Self-Ask (Press et al., 2023). Direct prompting asks the model to answer the original question, zero-shot CoT prompts it to reason through intermediate steps, and Self-Ask prompts it to generate and answer follow-up questions. We also evaluate a few-shot POKE variant that uses in-context demonstrations (Brown et al., 2020) to guide plan generation without planner.

POKE Planner. The planner is initialized from Qwen3-1.7B and trained using LoRA supervised fine-tuning. Given an original two-hop question, it generates a structured plan $\mathcal{P} = \langle A, O, L \rangle$, where A denotes the knowledge modules, O defines the required order constraints, and L specifies the dependency links used to propagate intermediate outputs between modules. The training target contains only the plan structure and does not reveal either the gold bridge entity or the final answer.

4.2 Knowledge Possession Does Not Guarantee Correct Reasoning

We first investigate a fundamental question: if an LLM already possesses all atomic knowledge required by a compositional problem, can it reliably organize and use that knowledge to derive the correct answer? To separate knowledge absence from knowledge-organization failure, we evaluate each model only on its own knowledge-known subset. Specifically, a two-hop question is included in S_{11}^{M} only when model M correctly answers both of its atomic questions independently. Therefore, failure on the composed question cannot be attributed to the absence of either required atomic fact.

Figure 2 provides an illustration of this setting. The planner first constructs the required knowledge actions together with their ordering constraint and dependency link. The executor then resolves the first action and binds its output to the dependent action before continuing execution. Thus, POKE does not provide the executor with new knowledge; instead, it specifies how knowledge already available to the model should be connected and used. To distinguish knowledge absence from knowledge-organization failure, we evaluate each model only on questions for which all constituent atomic facts can already be answered correctly by that model. We conducted this controlled evaluation on the three datasets. Although the datasets differ substantially in linguistic and compositional complexity, they exhibit the same phenomenon: having all required knowledge does not guarantee that the model can correctly organize and utilize it to answer the complete question. As shown in Table 1, conventional inference methods remain far from perfect even though every evaluated question contains only atomic facts that the model can already answer correctly. In contrast, POKE consistently achieves the best performance across all datasets and all three Qwen executors. Since POKE

Table 1: Results on each Qwen model's own knowledge-known subset for TwoHopFact, MuSiQue 2-hop and Wiki. Although all required atomic knowledge is already available, conventional inference methods still frequently fail, whereas POKE consistently achieves the highest accuracy.

Dataset	Method	Qwen3-1.7B	Qwen3-4B	Qwen3-8B
TwoHopFact	S_{11} examples	915	2,002	3,744
	Direct	75.41%	65.73%	64.85%
	Zero-shot CoT	77.92%	83.27%	85.39%
	Self-Ask	72.46%	76.87%	67.65%
	POKE	**98.47%**	**98.90%**	**99.07%**
MuSiQue 2-hop	S_{11} examples	7,155	11,227	11,920
	Direct	72.52%	72.16%	74.57%
	Zero-shot CoT	71.99%	69.96%	71.72%
	Self-Ask	69.85%	79.40%	78.01%
	POKE	**86.25%**	**91.45%**	**92.87%**
2WikiMultiHopQA (Wiki)	S_{11} examples	2,235	3,025	3,275
	Direct	83.09%	87.17%	83.45%
	Zero-shot CoT	84.56%	85.72%	86.72%
	Self-Ask	81.21%	87.40%	89.40%
	POKE	**94.36%**	**94.38%**	**94.53%**

Table 2: Answer accuracy on the 377 TwoHopFact examples for which all three models possess the required atomic knowledge. Qwen3-1.7B with POKE outperforms Qwen3-8B using other methods.

Method	Qwen3-1.7B	Qwen3-4B	Qwen3-8B
Direct	75.07%	79.05%	85.94%
Zero-shot CoT	83.02%	92.57%	96.02%
Self-Ask	73.47%	87.80%	87.80%
POKE	**98.14%**	**98.67%**	**99.73%**

neither introduces additional task knowledge nor updates the execution model, these improvements come from explicitly organizing and executing the knowledge that is already available. The results therefore demonstrate a consistent gap between knowledge possession and knowledge utilization, and show that explicit planning can substantially reduce this gap.

Knowledge organization can compensate for model scale. To control differences in knowledge coverage, we further evaluate three Qwen models on the same 377 TwoHopFact examples for which every model possesses both required atomic facts. As shown in Table 2, POKE enables Qwen3-1.7B to outperform the larger Qwen3-8B model using Direct, zero-shot CoT and Self-Ask, even though both models are evaluated on exactly the same questions and have been verified to possess all required atomic knowledge. More importantly, under Direct inference, Qwen3-1.7B and Qwen3-4B achieve very similar accuracy on this controlled subset. This indicates that once the availability of the required knowledge is controlled, increasing model scale alone provides little additional advantage in organizing and composing that knowledge in this setting. Together with the substantially larger S_{11} set of Qwen3-4B, these results suggest that a major benefit of model scaling lies in expanding knowledge coverage and representational capacity, while POKE targets a different source of improvement by explicitly organizing and utilizing knowledge that the model already possesses. The same pattern extends to MuSiQue and 2WikiMultiHopQA, full results are reported in Appendix B.

Few-shot planning. The above results use a trained planner to explicitly organize knowledge before execution. We further ask whether this organization pattern can be induced without planner fine-tuning. We find that a small number of in-context demonstrations is already sufficient to produce useful POKE plans and improve knowledge utilization on each model's own TwoHopFact S_{11} subset, suggesting that the structural principle of POKE is not restricted to a particular trained plan-

Table 3: Answer accuracy across different dependency topologies on each Qwen model's own MuSiQue knowledge-known subset. POKE achieves the strongest performance on most chain and join structures, while join structures remain substantially more challenging than chains.

Model	Method	C3	J3	C4	J4
Qwen3-1.7B	N	1,155	209	136	106
	Direct	51.43%	30.14%	47.06%	37.74%
	Zero-shot CoT	54.29%	30.14%	47.79%	41.51%
	Self-Ask	52.64%	29.19%	43.38%	40.57%
	POKE	**77.66%**	**48.80%**	**79.41%**	**59.43%**
Qwen3-4B	N	2,266	374	314	265
	Direct	58.87%	40.11%	59.24%	56.98%
	Zero-shot CoT	58.91%	43.05%	63.06%	63.02%
	Self-Ask	75.02%	65.24%	75.80%	78.49%
	POKE	**86.80%**	**67.65%**	**85.99%**	**80.38%**
Qwen3-8B	N	2,534	364	349	310
	Direct	59.67%	39.56%	66.19%	46.13%
	Zero-shot CoT	58.05%	39.84%	59.31%	44.84%
	Self-Ask	73.17%	60.99%	73.93%	62.26%
	POKE	**86.82%**	**62.91%**	**85.67%**	**63.87%**

ner. Nevertheless, the few-shot planner remains less reliable than the trained planner and shows sensitivity to demonstration selection. Full results are reported in Appendix C.

4.3 POKE Remains Effective for Complex Knowledge Dependencies

TwoHopFact and the two-hop evaluations primarily examine relatively short compositional dependencies. We next investigate whether POKE remains effective when both the depth and topology of the required knowledge dependencies become more complex. We used the three-hop and four-hop subsets of MuSiQue and evaluated each Qwen executor only on questions for which all constituent atomic probes have already been answered correctly. Rather than reporting only aggregate accuracy on all 3-4 hop questions, we analyze performance according to the underlying dependency topology. We distinguish three-hop chains (C3), three-hop joins (J3), four-hop chains (C4), and four-hop joins (J4). Chain structures propagate knowledge along a single dependency path, whereas join structures require multiple independent prerequisite branches to be resolved and subsequently integrated. The latter therefore provide a more direct test of the partial-order organization targeted by POKE. As shown in Table 3, increasing the depth of reasoning alone does not substantially degrade POKE. Its performance on C4 remains comparable to that on C3 in all three executors, despite the additional reasoning step. In contrast, join structures are consistently more difficult for most methods, indicating that dependency topology is a more important source of difficulty than chain length alone. POKE achieves the strongest performance on all evaluated topology combinations, with particularly large gains on C3 and C4. The remaining difficulty is concentrated in multi-branch dependency integration. Although POKE substantially improves J3 performance, its accuracy remains below that on the corresponding chain structures. The aggregate results for 3-4 hop questions, together with a gold-plan oracle analysis that separates planning errors from execution errors, are reported in Appendix D. We further decompose J4 into early and late-join structures in Appendix E.

4.4 POKE Generalizes Across Model Families

The previous experiments used Qwen3 as the primary execution-model family. We next investigate whether the benefit of POKE depends on the internal characteristics of Qwen or represents a more general inference-time knowledge organization mechanism. We evaluated POKE on several additional models from the DeepSeek-R1 and GLM families. As in previous experiments, each model is evaluated on its own S_{11} subset, ensuring that all required atomic knowledge is already available before comparing different inference strategies. Table 4 demonstrates that the effect of POKE is not

Table 4: TwoHopFact results on additional execution-model families. POKE consistently improves the utilization of already available knowledge across different models.

Model	Direct	CoT	Self-Ask	POKE
DeepSeek-R1-0528-Qwen3-8B	50.50%	72.17%	31.57%	**90.27%**
DeepSeek-R1-Distill-Qwen-1.5B	46.51%	57.67%	41.86%	**76.28%**
DeepSeek-R1-Distill-Qwen-7B	46.74%	81.39%	36.55%	**95.65%**
GLM-4-9B-0414	55.16%	79.21%	52.33%	**98.41%**

restricted to the Qwen3 models used in our main experiments. Across DeepSeek-R1-based executors and GLM-4, POKE consistently achieves substantially higher accuracy than other methods.

4.5 POKE for Physics-Based Simulation Script Generation

While the previous experiments study knowledge organization in multi-hop reasoning, we next examine whether the same principle can be extended to structured code generation. We apply POKE to the generation of scripting-language programs for physics-based simulation, where correctness depends not only on syntax but also on the organization of physical models, computational domains, boundary conditions, material properties, solver settings, and their dependencies. For this task, we extend the classical POKE representation to a simulation-specific plan: $P_{\mathrm{sim}} = \langle A, O, L, B, C, \Theta \rangle$, where B, C, and Θ additionally represent object/variable bindings, containment constraints, and physical or numerical parameter assignments, respectively. Physics-based simulation scripts provide a natural setting for evaluating partial-order knowledge organization. Pretrained LLMs often already possess substantial knowledge of simulation scripting languages, including syntax, common physical models, boundary conditions, solver options, and typical configuration patterns.

We evaluate Direct generation and POKE on 899 simulation-script generation examples manually constructed by domain experts in physics-based simulation. POKE improves all four structural and parameter-level metrics: OptAcc increases from 0.832 to 0.901, ChildCons from 0.803 to 0.884, FieldF1 from 0.846 to 0.923, and the more restrictive Hard metric from 0.725 to 0.869. These results suggest that explicit knowledge organization is also useful when the output itself is a structured program whose modules must satisfy multiple interdependent constraints. The detailed task description, metric definitions, and a representative POKE generation example are provided in Appendix F.

4.6 Ablation and inference efficiency

We also conduct controlled ablations to probe the roles of dependency propagation, execution order, and stepwise execution. Disrupting intermediate bindings or reversing dependency order causes substantial performance degradation, while replacing the predicted bridge with the gold entity yields nearly perfect accuracy. We further compare model calls, generation latency, and token consumption across inference strategies. POKE achieves the highest accuracy while requiring fewer calls and substantially fewer tokens than repeated CoT and Self-Ask, resulting in a favorable accuracy cost trade-off. The complete results are reported in Appendices G and H.

5 Conclusion

We introduce **Partial Order Knowledge Engineering (POKE)**, a planning-based framework that organizes knowledge modules, ordering constraints, and causal links for an LLM executor. Evaluations on atomic-probe subsets of TwoHopFact, MuSiQue and 2WikiMultiHopQA show improved compositional accuracy over conventional prompting. On the shared TwoHopFact subset, Qwen3-1.7B with POKE outperforms Qwen3-8B with other conventional baselines, demonstrating the value of explicit knowledge organization for smaller executors. Experiments on longer questions further show gains across most evaluated chain and join settings. We also extend POKE to physics-based simulation script generation by incorporating object bindings, hierarchical containment, and physical and numerical parameters into the plan.

AI USE STATEMENT

AI-assisted tools were used solely for grammar correction, typo checking, and minor language polishing during manuscript preparation. They were not used to generate research ideas, conduct experiments, analyze results, or make scientific conclusions.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the datasets, model-specific knowledge subset construction, evaluation protocols, prompting strategies, planner training procedure, and implementation settings used in our experiments. For each knowledge-known subset, we specify the atomic probing procedure and the exact inclusion criterion, enabling the evaluation subsets to be reconstructed from the original datasets and model outputs. We also report the planner architecture, LoRA configuration, plan representation, execution procedure, evaluation metrics, sample sizes, model-call counts, generation latency, and token consumption.

The prompts and additional experimental details, including ablations, topology-specific analyses, few-shot planning, and oracle evaluations, are provided in the appendix or supplementary material. Code, evaluation scripts, prompts, and additional implementation details will be made available where permitted by the licenses of the corresponding models and datasets.

REFERENCES

Mohamed Aghzal, Erion Plaku, Gregory J. Stein, and Ziyu Yao. A survey on large language models for automated planning, 2025. URL https://arxiv.org/abs/2502.12435.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, and Girish Sastry. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Yuxuan Cai, Jiangning Zhang, Haoyang He, Xinwei He, Ao Tong, Zhenye Gan, Chengjie Wang, Zhucun Xue, Yong Liu, and Xiang Bai. Llava-kd: A framework of distilling multimodal large language models, 2025. URL https://arxiv.org/abs/2410.16236.

Pengfei Cao, Tianyi Men, Wencan Liu, Jingwen Zhang, Xuzhao Li, Xixun Lin, Dianbo Sui, Yanan Cao, Kang Liu, and Jun Zhao. Large language models for planning: A comprehensive and systematic survey, 2025. URL https://arxiv.org/abs/2505.19683.

F. de Saussure, C. Bally, and A. Sechehaye. *Course in General Linguistics*. P. Owen, 1960. URL https://books.google.com.hk/books?id=SPotwgEACAAJ.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023. URL https://arxiv.org/abs/2211.10435.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, and Guanyu Feng. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024. URL https://arxiv.org/abs/2406.12793.

Guoqiang Gong, Jiaxing Wang, Jin Xu, Pengzhang Liu, and Ke Zhang. Beyond logits: Aligning feature dynamics for effective knowledge distillation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23067–23077, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1125. URL https://aclanthology.org/2025.acl-long.1125/.

Daya Guo, Dejian Yang, Haowei Zhang, and Junxiao Song. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps, 2020. URL https://arxiv.org/abs/2011.01060.

Tao Huang, Shan You, Fei Wang, Chen Qian, and Chang Xu. Knowledge distillation from a stronger teacher, 2022a. URL https://arxiv.org/abs/2205.10536.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022b. URL https://arxiv.org/abs/2201.07207.

Minki Kang, Seanie Lee, Jinheon Baek, Kenji Kawaguchi, and Sung Ju Hwang. Knowledge-augmented reasoning distillation for small language models in knowledge-intensive tasks, 2023. URL https://arxiv.org/abs/2305.18395.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity, 2022. URL https://arxiv.org/abs/2104.08786.

Jiaming Lv, Haoyuan Yang, and Peihua Li. Wasserstein distance rivals kullback-leibler divergence for knowledge distillation, 2024. URL https://arxiv.org/abs/2412.08139.

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant, 2019. URL https://arxiv.org/abs/1902.03393.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, and Carroll L. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation, 2019. URL https://arxiv.org/abs/1904.05068.

J. Scott Penberthy and Daniel S. Weld. Ucpop: A sound, complete, partial order planner for adl. In *International Conference on Principles of Knowledge Representation and Reasoning*, 1992. URL https://api.semanticscholar.org/CorpusID:5329862.

Mary Phuong and Christoph H. Lampert. Towards understanding knowledge distillation, 2021. URL https://arxiv.org/abs/2105.13093.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023. URL https://arxiv.org/abs/2210.03350.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model, 2025. URL https://arxiv.org/abs/2405.14205.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models, 2023. URL https://arxiv.org/abs/2305.16653.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, and Lukas Blecher. Llama 2: Open foundation and fine-tuned chat models, 2023b. URL https://arxiv.org/abs/2307.09288.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation, 2019. URL https://arxiv.org/abs/1907.09682.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change, 2023. URL https://arxiv.org/abs/2206.10498.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models, 2023a. URL https://arxiv.org/abs/2305.04091.

Peifeng Wang, Zhengyang Wang, Zheng Li, Yifan Gao, Bing Yin, and Xiang Ren. Scott: Self-consistent chain-of-thought distillation, 2023b. URL https://arxiv.org/abs/2305.01879.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020. URL https://arxiv.org/abs/2002.10957.

Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting, 2024. URL https://arxiv.org/abs/2402.10200.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023c. URL https://arxiv.org/abs/2203.11171.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, 2024. URL https://arxiv.org/abs/2302.01560.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Daniel S. Weld. An introduction to least commitment planning. *AI Mag.*, 15:27–61, 1994. URL https://api.semanticscholar.org/CorpusID:721902.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with jacobi iteration, 2026. URL https://arxiv.org/abs/2506.18582.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Self-evaluation guided beam search for reasoning, 2023. URL https://arxiv.org/abs/2305.00633.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. Faithful logical reasoning via symbolic chain-of-thought, 2024. URL https://arxiv.org/abs/2405.18357.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, and Binyuan Hui. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning?, 2025b. URL https://arxiv.org/abs/2402.16837.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Meiru Zhang, Zaiqiao Meng, and Nigel Collier. Failure modes in multi-hop qa: The weakest link effect and the recognition bottleneck, 2026. URL https://arxiv.org/abs/2601.12499.

Weijia Zhang, Fei Xie, Weidong Cai, and Chao Ma. Vrm: Knowledge distillation via virtual relation matching, 2025. URL https://arxiv.org/abs/2502.20760.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models, 2022. URL https://arxiv.org/abs/2210.03493.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2024a. URL https://arxiv.org/abs/2310.04406.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023. URL https://arxiv.org/abs/2205.10625.

Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, and Huaixiu Steven Zheng. Self-discover: Large language models self-compose reasoning structures, 2024b. URL https://arxiv.org/abs/2402.03620.

A Dataset Details and Evaluation Construction

This section provides additional details on the three multi-hop question-answering datasets used in our experiments and explains how each dataset is incorporated into the evaluation of POKE. Rather than treating all questions as equally informative about knowledge organization, we distinguish between the original compositional question and the atomic knowledge required to solve it. This allows us to evaluate whether an executor can correctly compose knowledge that it has already demonstrated the ability to retrieve independently.

A.1 TwoHopFact

TwoHopFact is our primary controlled dataset for studying the gap between atomic knowledge retrieval and compositional reasoning. Each example has an explicit two-hop relational structure

$$e_2 = r_1(e_1), \qquad e_3 = r_2(e_2), \tag{10}$$

where e_1 is the source entity, e_2 is the intermediate bridge entity, and e_3 is the final answer. The corresponding compositional question therefore requires two knowledge operations: first recovering e_2 from e_1, and then using e_2 as the argument required to recover e_3.

This explicit decomposition makes TwoHopFact particularly suitable for controlled knowledge probing. For executor M, we independently query the two constituent atomic questions and define

$$S_{11}^M = \{q : \mathrm{Correct}_M(q_1) = 1 \wedge \mathrm{Correct}_M(q_2) = 1\}. \tag{11}$$

Membership in S_{11}^M means that the executor successfully answered both constituent probes under the evaluation protocol. We then evaluate the original two-hop question on the same subset. Consequently, an incorrect compositional answer on S_{11}^M cannot be directly attributed to failure on either independently evaluated atomic probe.

TwoHopFact is used in four parts of our experiments. First, the model-specific S_{11}^M subsets provide the primary controlled evaluation of the knowledge-organization gap in Section 4.2. Second, the intersection

$$S_{11}^{\mathrm{shared}} = \bigcap_{M \in \mathcal{E}} S_{11}^M \tag{12}$$

is used for controlled comparisons across executor scales, ensuring that all compared models are evaluated on exactly the same questions and have successfully answered the same atomic probes. Third, TwoHopFact is used for controlled ablations of intermediate-result binding, dependency propagation, execution order, and stepwise execution. Finally, it is used for the few-shot planning study and for evaluation across additional executor families.

Importantly, S_{11}^M is an operational knowledge-probing criterion rather than a claim about permanent internal knowledge representation. It establishes that the model successfully retrieves the required atomic components under the specified probing protocol, which is sufficient for our controlled comparison between atomic retrieval and compositional use.

A.2 MuSiQue

MuSiQue provides a substantially more diverse setting for evaluating multi-hop knowledge composition. In contrast to the relatively regular two-step relational structure of TwoHopFact, MuSiQue contains natural-language questions constructed from multiple constituent questions and supports reasoning structures with different numbers of hops and dependency topologies. We use the two-hop portion of MuSiQue as a second dataset for the controlled knowledge-organization experiment. Analogous to TwoHopFact, an executor is evaluated only on questions for which the required atomic probes have been answered correctly, producing a model-specific S_{11}^M subset. This allows us to test whether the knowledge-organization gap observed on TwoHopFact also appears in a more diverse natural-language setting.

We additionally use three-hop and four-hop MuSiQue questions to evaluate POKE under longer and structurally more complex dependencies. For these questions, we extend the atomic-probe criterion to

$$S_{111}^{M} = \left\{ q : \bigwedge_{i=1}^{3} \mathrm{Correct}_M(q_i) = 1 \right\}, \tag{13}$$

$$S_{1111}^{M} = \left\{ q : \bigwedge_{i=1}^{4} \mathrm{Correct}_M(q_i) = 1 \right\}. \tag{14}$$

These subsets ensure that all constituent probes associated with the corresponding three-hop or four-hop question have been answered correctly before evaluating the complete compositional problem. Beyond hop count, MuSiQue also enables us to study dependency topology. We divide the evaluated questions into four principal structures:

- **C3**: three-hop chain;
- **J3**: three-hop structure containing a join;
- **C4**: four-hop chain;
- **J4**: four-hop structure containing a join.

Chain structures predominantly require knowledge to be propagated along a single dependency path. Join structures instead contain multiple prerequisite branches whose intermediate outputs must be resolved and combined before a downstream module can be executed. This distinction makes MuSiQue especially important for evaluating the partial-order aspect of POKE: independent branches can remain unordered until their outputs are required by a shared downstream module.

MuSiQue therefore serves three complementary purposes in our experiments. Its two-hop subset tests whether the knowledge-organization gap generalizes beyond TwoHopFact; its three- and four-hop subsets evaluate robustness to increasing reasoning depth; and its chain/join categorization evaluates sensitivity to dependency topology. We further use gold dependency plans on the three- and four-hop subsets to distinguish errors arising from plan construction from those arising during plan execution.

A.3 WIKI

Wiki provides an additional large-scale multi-hop evaluation setting. Our current evaluation cohort contains 4,914 compositional questions. Unlike the primary TwoHopFact diagnostic, Wiki is used primarily to test whether the observed knowledge-organization effects extend to an additional multi-hop cohort and whether the cross-scale findings persist on a second shared evaluation set.

For each executor M, we construct a model-specific atomic-probe subset S_{11}^{M} according to the Wiki evaluation protocol. The resulting subset sizes are 2,235 questions for Qwen3-1.7B, 3,025 for Qwen3-4B, and 3,275 for Qwen3-8B. Because these subsets contain different questions for different executors, they are used only to compare inference methods within the same executor, rather than to compare absolute performance across model scales. On these subsets, POKE achieves answer accuracy of 94.36%, 94.38%, and 94.53% for Qwen3-1.7B, Qwen3-4B and Qwen3-8B, respectively. These experiments provide a third source of evidence that successful atomic probing does not necessarily translate into reliable compositional answering, and that explicit knowledge organization can improve conditional compositional accuracy.

B SHARED KNOWLEDGE-KNOWN EVALUATION

To complement the model-specific evaluations in the main text, we construct shared knowledge-known subsets on MuSiQue and 2WikiMultiHopQA. A question is included only when all three Qwen executors successfully answer the required atomic probes under the corresponding dataset

Table 5: Answer accuracy on the shared knowledge-known subsets of MuSiQue and 2WikiMultiHopQA. All three Qwen executors are evaluated on identical questions within each dataset.

Dataset	Method	Qwen3-1.7B	Qwen3-4B	Qwen3-8B
MuSiQue(C2)	Shared examples	5,911	5,911	5,911
	Direct	75.54%	77.72%	82.20%
	Zero-shot CoT	75.11%	75.40%	78.99%
	Self-Ask	71.97%	84.96%	84.50%
	POKE	**87.97%**	**93.83%**	**95.31%**
2WikiMultiHopQA	Shared examples	1,934	1,934	1,934
	Direct	85.88%	92.86%	91.05%
	Zero-shot CoT	87.23%	91.78%	93.23%
	Self-Ask	84.38%	92.97%	94.42%
	POKE	**95.24%**	**95.29%**	**95.35%**

Table 6: Few-shot POKE results on each Qwen model's own TwoHopFact S_{11} subset. Few-shot demonstrations can induce useful POKE plans without planner fine-tuning, while the trained planner remains substantially more reliable.

Planner	Qwen3-1.7B	Qwen3-4B	Qwen3-8B
S_{11} examples	915	2,002	3,744
1-shot planner	87.32%	85.41%	84.75%
3-shot planner	90.16%	84.47%	82.88%
5-shot planner	91.69%	88.66%	85.76%
Trained POKE planner	**98.47%**	**98.90%**	**99.07%**

protocol. Therefore, all models are evaluated on exactly the same questions within each dataset, reducing the effect of differences in atomic knowledge coverage.

As shown in Table 5, POKE achieves the highest answer accuracy for every executor on both shared subsets. On MuSiQue, POKE improves over the strongest reported baseline by 12.43, 8.86 and 10.81 percentage points for Qwen3-1.7B, Qwen3-4B and Qwen3-8B, respectively. On 2WikiMultiHopQA, the corresponding improvements are 8.01, 2.33, and 0.93 percentage points. The shared-subset results also provide an additional cross-scale comparison. On MuSiQue, Qwen3-1.7B with POKE achieves 87.97%, compared with 84.50% for Qwen3-8B with Self-Ask, the strongest baseline for the 8B executor. Similarly, on 2WikiMultiHopQA, Qwen3-1.7B with POKE achieves 95.24%, compared with 94.42% for Qwen3-8B with Self-Ask. Together with the TwoHopFact shared-subset results in the main text, these findings show that explicit knowledge organization can compensate for a substantial portion of the advantage associated with increasing executor scale when atomic-probe success is controlled.

C Few-Shot Planning Without Planner Training

The main experiments use a trained POKE planner to generate structured partial-order plans. Here, we investigate whether the POKE planning pattern can also be induced without additional planner training. Instead of fine-tuning a dedicated planner, we provide the base Qwen3 model with a small number of demonstrations illustrating how to decompose a compositional question into knowledge modules and specify their dependencies and execution order. This setting isolates whether the structural principle of POKE can be learned directly through in-context examples rather than being tied to a particular fine-tuned planner.

As shown in Table 6, few-shot demonstrations are sufficient to induce a useful POKE planning pattern without parameter updates. On Qwen3-1.7B, performance improves from 87.32% with one demonstration to 91.69% with five demonstrations. For Qwen3-4B and Qwen3-8B, however, performance is not monotonic in the number of demonstrations: the 3-shot setting underperforms the 1-shot setting, while 5-shot planning recovers to 88.66% and 85.76%, respectively. This sensitivity

Table 7: Aggregate answer accuracy on each Qwen model's own MuSiQue three-hop and four-hop knowledge-known subset. Gold-plan POKE replaces the learned plan with the gold dependency structure and therefore serves as an oracle upper bound rather than a deployable inference method.

Method	**Qwen3-1.7B**	**Qwen3-4B**	**Qwen3-8B**
Examples	1,606	3,219	3,557
Direct	47.38%	56.57%	57.07%
Zero-shot CoT	49.75%	57.81%	55.16%
Self-Ask	48.01%	74.25%	71.04%
POKE	**72.85%**	**83.97%**	**82.26%**
Gold-plan POKE	93.09%	94.63%	96.49%

indicates that in-context planning depends not only on the number of demonstrations but also on how effectively those demonstrations induce the intended dependency structure.

The trained planner remains substantially stronger, reaching 98.47%, 98.90%, and 99.07% on the three model-specific S_{11} subsets. These results show that the POKE planning pattern can be induced through prompting alone, but dedicated planner training provides substantially more reliable knowledge decomposition and dependency construction.

D Aggregate Multi-Hop and Gold-Plan Results

In the main text, we analyze MuSiQue three-hop and four-hop questions according to their dependency topology. Here, we additionally report aggregate results over all model-specific S^M_{111} and S^M_{1111} examples and examine the performance upper bound obtained when the learned planner is replaced by the gold dependency plan. Table 7 confirms the topology-specific results in the main text at the aggregate level. POKE improves substantially over Direct, zero-shot CoT, and Self-Ask on the combined three- and four-hop knowledge-known subsets for all three executors. To distinguish planning errors from execution errors, we additionally replace the learned plan with the gold dependency structure while keeping the executor unchanged. This increases answer accuracy from 72.85% to 93.09% for Qwen3-1.7B, from 83.97% to 94.63% for Qwen3-4B, and from 82.26% to 96.49% for Qwen3-8B, corresponding to gains of 20.24, 10.66, and 14.23 percentage points, respectively.

The large gap between learned-plan and gold-plan execution indicates that a substantial fraction of the remaining errors arises during plan construction. Once the correct dependency structure is supplied, the same executors can solve a much larger fraction of the questions. Together with the lower performance on join structures reported in the main text, this result identifies dependency construction and multi-branch integration as major remaining bottlenecks of the current POKE planner.

E Detailed Analysis of Dependency Topology

We provide additional topology-specific results for the 4-hop join structures in MuSiQue. In particular, we further divide J4 into early-join and late-join structures according to the position at which multiple knowledge branches are merged.

As shown in Table 8, the finer-grained results further support the observation in Section 4.3 that dependency topology is an important source of difficulty. POKE achieves the best performance on both J4-Early and J4-Late for all three Qwen executors. For Qwen3-1.7B and Qwen3-8B, performance decreases from early joins to late joins, indicating that delaying the integration of multiple prerequisite branches can make reasoning more difficult. Qwen3-4B shows a different pattern, with comparable or slightly better performance on J4-Late, suggesting that the effect of join position is model dependent. Overall, the results show that POKE remains effective on both early- and late-join structures, while multi-branch dependency integration remains more challenging than simple chain reasoning.

Table 8: Detailed results for early- and late-join 4-hop structures on each model's own knowledge-known subset.

Model	Method	J4-Early		J4-Late	
		N	Acc.	N	Acc.
Qwen3-1.7B	Direct	52	46.15%	54	29.63%
	Zero-shot CoT	52	50.00%	54	33.33%
	Self-Ask	52	44.23%	54	37.04%
	POKE	52	**63.46%**	54	**55.56%**
Qwen3-4B	Direct	74	63.51%	191	54.45%
	Zero-shot CoT	74	64.86%	191	62.30%
	Self-Ask	74	72.97%	191	80.63%
	POKE	74	**78.38%**	191	**81.15%**
Qwen3-8B	Direct	92	68.48%	218	36.70%
	Zero-shot CoT	92	66.30%	218	35.78%
	Self-Ask	92	79.35%	218	55.05%
	POKE	92	**80.43%**	218	**56.88%**

F Physics-Based Simulation Script Generation

We provide additional details on the physics-based simulation script generation experiment introduced in Section 4.5. All experiments in this section use Qwen3-1.7B as the execution model. We evaluate Direct generation and POKE on 899 simulation-script generation examples using the same executor, so that the comparison isolates the effect of explicit knowledge organization rather than model scale.

Unlike multi-hop question answering, this task requires the model to generate a structured simulation script in which physical components, hierarchical blocks, object references, and numerical parameters must satisfy multiple constraints simultaneously.

F.1 Task and POKE Representation

Given a natural-language simulation specification, the model is required to generate the corresponding simulation script. The specification may describe material properties, analysis type, physical models, computational domains, boundary conditions, solver configurations, and numerical parameters. Correct generation therefore depends not only on selecting the appropriate individual configuration fields, but also on organizing them under the correct objects and hierarchical blocks.

For this setting, we extend the standard POKE representation to

$$\mathcal{P}_{\text{sim}} = \langle A, O, L, B, C, \Theta \rangle,$$

where A contains the simulation configuration modules, O specifies ordering constraints, and L describes dependency relations between modules. B specifies object and variable bindings, C represents hierarchical containment constraints, and Θ stores the physical and numerical parameter assignments.

The resulting plan explicitly separates knowledge about *what* must be configured from knowledge about *where*, *in what dependency context*, and *with which parameter values* it should appear in the generated script.

F.2 Quantitative Results

We evaluate Direct generation and POKE on 899 simulation-script generation examples. We report four complementary metrics that measure different levels of structural and parameter consistency.

Table 9: Simulation-script generation results on 899 examples using Qwen3-1.7B as the executor. OptAcc measures whether the decoded root option matches the required conditioned decision; ChildCons measures exact consistency of the enabled child blocks; FieldF1 evaluates required numeric fields implied by the selected option; Hard evaluates ChildCons on the more restrictive `INLET` and `FLOW DIRECTION` cases. Higher is better for all metrics.

Method	OptAcc	ChildCons	FieldF1	Hard
Direct	0.832	0.803	0.846	0.725
POKE	**0.901**	**0.884**	**0.923**	**0.869**

As shown in Table 9, using the same Qwen3-1.7B executor, POKE improves all four metrics over Direct generation, showing that the gains arise from the structured planning and organization procedure rather than from using a larger execution model. OptAcc increases by 6.9 percentage points, indicating more reliable selection of the required configuration option. ChildCons improves by 8.1 points, showing that POKE more frequently generates the complete set of child blocks implied by the selected configuration. FieldF1 increases by 7.7 points, suggesting improved assignment of required numerical fields. The largest improvement appears on the Hard subset, where accuracy increases from 0.725 to 0.869, a gain of 14.4 percentage points. These cases place stronger requirements on the joint consistency of boundary configuration and flow-direction structure. The larger improvement therefore suggests that the benefit of POKE becomes particularly pronounced when multiple configuration decisions must satisfy coupled structural constraints.

F.3 Representative Generation Example

We further illustrate how POKE organizes a representative physics-based simulation request. The input specifies a steady-state laminar air-flow simulation with prescribed material properties, a stationary non-buoyant fluid domain, spatially varying inlet velocity data, two zero-pressure outlets, a fixed no-slip wall, and a symmetry boundary.

The planner first converts the specification into structured configuration modules. A simplified view of the resulting plan is

$$A = \{\texttt{Material(Air)}, \texttt{SteadyState}, \texttt{FluidDomain}, \texttt{Laminar}, \texttt{Inlet}, \texttt{Outlet}_1, \texttt{Outlet}_2, \texttt{Wall}, \texttt{Symmetry}, \ldots\}. \tag{15}$$

The plan then specifies relations such as

$$\texttt{Material} \prec \texttt{MaterialProperties}, \qquad \texttt{Domain} \prec \texttt{Boundary},$$

together with causal and containment relations that associate the material with the fluid definition and place the inlet, outlets, wall, and symmetry conditions under the corresponding fluid domain. Parameter bindings Θ additionally instantiate quantities such as density, specific heat, viscosity, pressure, and solver settings.

```
MATERIAL: Air
  PROPERTIES:
    EQUATION OF STATE:
      Density = 1 [kg m^-3]
    END
    SPECIFIC HEAT CAPACITY:
      Specific Heat Capacity = 1004.4 [J kg^-1 K^-1]
    END
    DYNAMIC VISCOSITY:
      Dynamic Viscosity = 0.003333333 [Pa s]
    END
  END
END
```

```
FLOW: Flow Analysis 1
  ANALYSIS TYPE:
    Option = Steady State
  END
  DOMAIN: Default Domain
    DOMAIN MODELS:
      BUOYANCY MODEL:
        Option = Non Buoyant
      END
      DOMAIN MOTION:
        Option = Stationary
      END
    END
    FLUID MODELS:
      TURBULENCE MODEL:
        Option = Laminar
      END
    END
    ...
  END
END
```

The excerpt shows how physical properties, analysis settings, domain models, and boundary configurations are instantiated under their corresponding hierarchical blocks. The full generated script contains the remaining boundary and solver configurations.

G Ablation Studies

We conduct controlled ablations to examine whether the gains of POKE arise from its explicit knowledge organization rather than simply from additional prompting or model calls. Specifically, we investigate four key components: intermediate-result propagation, dependency links, execution order, and sequential module execution. Each model is evaluated on its own TwoHopFact S_{11} subset, where both required atomic facts have been verified to be available to that model.

Full POKE executes the planned subquestions separately and propagates the resolved bridge entity to the downstream module (Table 10). Supplying the gold bridge yields nearly perfect accuracy, consistent with the second-hop success required for inclusion in S_{11}. This result provides an execution diagnostic under the subset selection protocol.

Source as bridge reduces accuracy by supplying the original source entity as the downstream argument. *Unresolved bridge* and *Reverse order* both execute the downstream query without a resolved bridge entity and produce identical aggregate accuracies. These interventions demonstrate sensitivity to missing or incorrect intermediate inputs, but do not separately quantify the contributions of ordering constraints and causal links.

Single-call execution retains the plan while changing call structure, context organization, and intermediate-result binding. Its lower accuracy supports the effectiveness of the full execution procedure in this comparison. Matched-budget comparisons with alternative decomposition and execution procedures are needed to isolate the contribution of the partial-order representation.

H Inference Cost and Efficiency Analysis

We further analyze the inference cost of POKE on each Qwen model's own TwoHopFact S_{11} subset. We compare POKE with single-sample and multi-sample versions of Direct prompting, zero-shot CoT, and Self-Ask, as well as few-shot POKE planners and the execution variants used in our ablation study. For the conventional prompting baselines, *1-sample* uses one sampled response per question. *Majority@5* generates five independent responses and returns the majority answer, while *Pass@5* uses the same five responses but counts the question as correct if any response is correct.

Table 10: Ablation results on each Qwen model's own TwoHopFact S_{11} subset. Each variant selectively modifies the planning or execution procedure. We report strict exact-match accuracy following the ablation protocol.

Setting	Description	1.7B	4B	8B
S_{11} examples	Number of model-specific knowledge-known questions.	915	2,002	3,744
Full POKE	A trained planner generates the plan, and the executor follows the planned dependencies step by step, executing A_1, resolving its output, and then executing A_2.	**98.47%**	**98.90%**	**99.07%**
Gold bridge	Replace the predicted intermediate entity with the gold bridge entity before executing the downstream module.	100.00%	99.90%	99.95%
Single-call execution	Generate the complete plan, but do not execute A_1 and A_2 separately. Instead, place the original question and the entire plan into a single prompt and ask the executor to produce the final answer in one call.	37.27%	77.52%	73.32%
Source as bridge	Execute A_1 normally, but deliberately ignore its output when instantiating A_2 and use the original source entity instead (e.g., *The Republic* $\rightarrow$ *Plato*, while A_2 still queries *The Republic* rather than *Plato*).	69.40%	64.34%	63.78%
Unresolved bridge	Execute A_1 normally, but leave the bridge placeholder in A_2 unresolved instead of replacing it with the predicted entity (e.g., "Where was *bridge_entity* born?" rather than "Where was *Plato* born?").	10.93%	7.59%	15.60%
Reverse order	Deliberately violate the dependency order: execute A_2 first with an unresolved *bridge_entity*, and execute A_1 only afterward, instead of resolving the bridge with A_1 before invoking A_2.	10.93%	7.59%	15.60%

Thus, Majority@5 and Pass@5 have identical inference cost, but Pass@5 is an oracle evaluation because selecting the successful response requires access to the gold answer.

Figure 3 jointly compares answer quality and inference cost. Panel (a) shows that POKE reaches 98.5%, 98.9% and 99.1% answer accuracy with the Qwen3-1.7B, Qwen3-4B and Qwen3-8B executors, respectively, substantially exceeding all non-oracle prompting baselines. Increasing the sampling budget alone does not close this gap: even Majority@5 remains well below POKE, while Pass@5 represents an oracle upper bound over the same five sampled responses. The improvement is not obtained by simply increasing the number of model calls. As shown in Panel (c), POKE requires three calls per question, one planner call and two executor calls, whereas Direct and CoT Majority@5 require five calls and Self-Ask Majority@5 requires ten. The ablation variants further show that most POKE configurations retain approximately the same call budget, allowing their accuracy differences to be attributed to how intermediate knowledge is organized and propagated rather than to additional model invocations. Panel (d) shows a similar pattern in token consumption. Full POKE uses only 477–480 tokens per question across the three executors, substantially fewer than CoT Majority@5 (799–870 tokens) and Self-Ask Majority@5 (1,077–1,173 tokens). In contrast, increasing the number of in-context planning demonstrations substantially increases token usage: the 5-shot planner requires more than 1,200 tokens per question while remaining less accurate than the trained POKE planner. This comparison highlights the benefit of learning a compact planner rather than repeatedly providing long planning demonstrations at inference time. The main additional cost of POKE is latency. As shown in Panel (b), POKE requires 3.67, 3.14, and 3.14 seconds per question for the three executors, respectively. This is slower than single-pass Direct prompting because an explicit planning stage is introduced, but substantially faster than CoT Majority@5, which requires

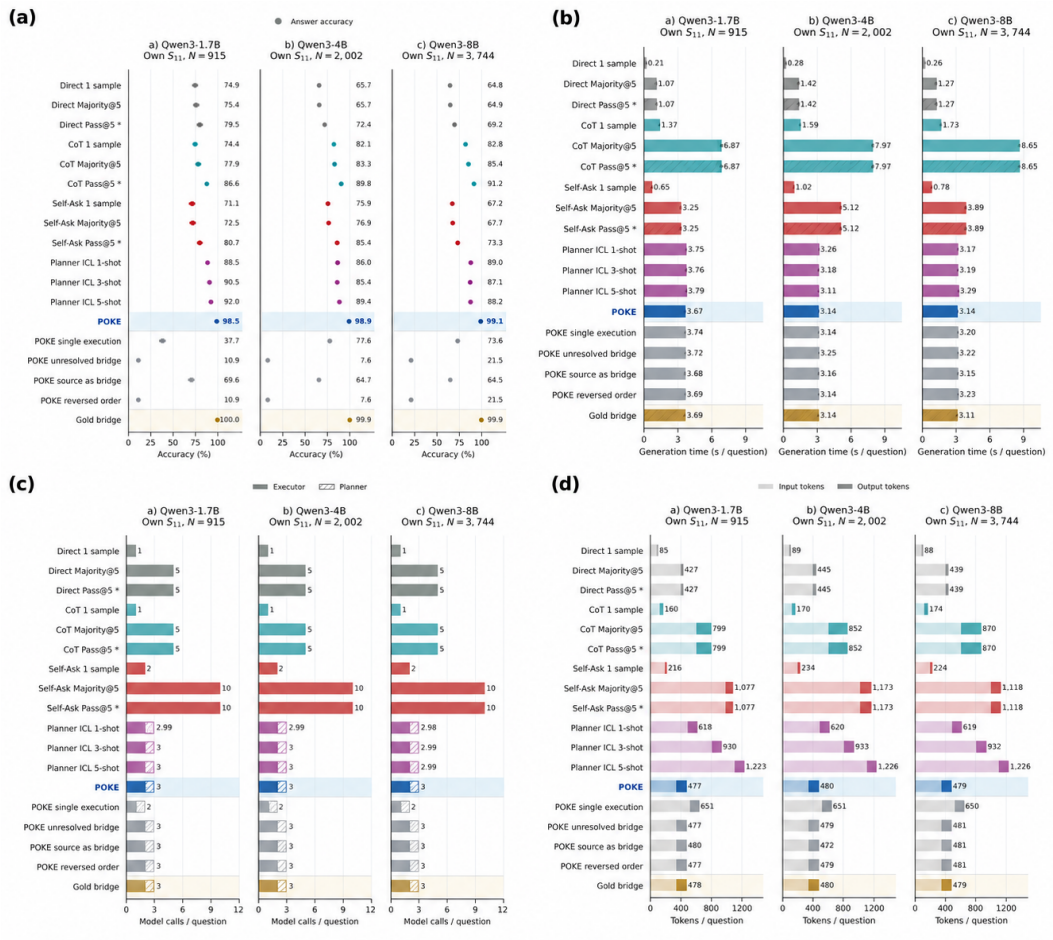


Figure 3: Accuracy and inference-cost analysis on each Qwen model's own TwoHopFact S_{11} subset. **(a)** Answer accuracy under different inference configurations; **(b)** Mean generation latency per question, summed over all model calls; **(c)** Number of model calls per question, with solid regions denoting executor calls and hatched regions denoting planner calls. **(d)** Input and output token consumption per question. POKE achieves the highest non-oracle accuracy with three model calls and approximately 477–480 tokens per question, while introducing moderate planning latency relative to single-pass Direct prompting.

6.87–8.65 seconds per question, and is comparable to or faster than Self-Ask Majority@5 for the larger executors.

Overall, the four measurements show that POKE does not obtain its performance gain simply by allocating a larger inference budget. Single-pass Direct prompting remains the cheapest strategy in absolute terms, but its accuracy is substantially lower. POKE instead achieves the strongest overall accuracy–cost trade-off among the evaluated non-oracle reasoning strategies: it reaches near-perfect accuracy with only three calls and approximately 480 tokens per question, without relying on repeated sampling or oracle answer selection.