Metacognition-Driven Cognitive Planning for Pedagogical Dialogue: A Case Study in Math Tutoring

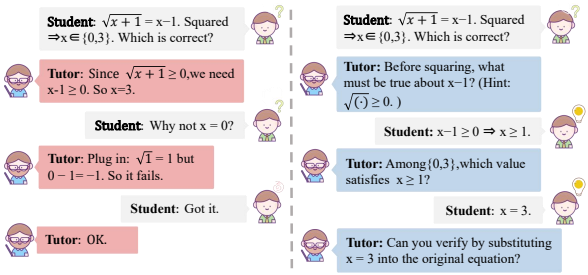
Anonymous ACL submission

Abstract

For complex cognitive planning problems, existing chain-of-thought prompting and agent-based methods lack a standardized problem-solving guidance framework, often leading to scattered planning logic, limited interpretability, and weak domain adaptation. We propose a metacognition-driven dynamic cognitive planning method that determines an instance-specific planning template based on the intrinsic attributes of the target problem, and uses this template to guide professional and trustworthy solving of complex planning tasks. Specifically, our method (i) performs deep decomposition to uncover the problem's intrinsic attributes, (ii) synthesizes a problem-specific cognitive planning template, (iii) generates a fine-grained planning trace following the template, and (iv) produces a standardized output derived from the trace. We validate the approach in educational math tutoring as a representative cognitive planning setting, covering diverse tasks on MathTutorBench including mathematical problem solving, student-solution diagnosis/understanding, and scaffolding-oriented tutoring. Experiments show that our method significantly outperforms strong prompting baselines and fine-tuned tutors, and substantially improves controllability and trace interpretability under strict output constraints and multi-turn interaction. These results suggest a new paradigm for solving complex cognitive planning problems via dynamic template synthesis and trace-based execution.

1 Introduction

Large language models (LLMs) show strong reasoning and generation abilities (Achiam et al., 2023; Touvron et al., 2023; Bai et al., 2023). This makes them promising for tasks that require multi-step decision making and structured outputs. Yet many high-stakes tasks require more than "getting an answer". They require a stable task understanding, a coherent plan, and an intermediate process that can be inspected, supervised, or reused.



(a) gives verdict and justification. (b) selects micro-goals and asks guiding questions.

Figure 1: **Tutoring is not just correctness.** A MATHTUTORBENCH tutoring snapshot contrasts answer-giving (novice) with Socratic scaffolding (expert), where the tutor selects micro-goals and elicits self-correction instead of directly revealing the final answer.

A typical failure of end-to-end prompting is *process instability*. Even when prompted to "think step by step" (Wei et al., 2022; Kojima et al., 2022), the model may jump between different intents, change granularity across steps, or overfit to surface instructions. In tutoring dialogue, this can appear as switching from diagnosis to direct solving, skipping key checks, or producing guidance that does not match the student's current state (cf. tutoring principles such as scaffolding and feedback; Anghileri, 2006; Hattie and Timperley, 2007). Figure 1 illustrates a standardized tutoring gap: two tutors may both "know" the mathematics, yet differ sharply in pedagogy—a good tutor plans micro-goals and guides self-correction, rather than immediately providing the verdict.

In some benchmarks, additional constraints exist (e.g., strict formats or "no direct answer"). These constraints matter in practice, but they are not the core difficulty. The core difficulty is that the model lacks a stable *cognitive planning process* that separates (i) task understanding, (ii) plan formation, and (iii) step-wise execution into a consistent structure. Moreover, recent evidence suggests that LLMs do not reliably self-correct their reasoning without ex-

plicit control mechanisms (Huang et al., 2023).

We view tutoring as a representative case of a broader class of *complex cognitive planning tasks*. In such tasks, the agent should: (1) form a panoramic understanding of goals, requirements, and bottlenecks, (2) build an executable plan structure that fits the current instance, (3) execute the plan while keeping intermediate results interpretable, and (4) derive the final output from that intermediate trace. This intermediate trace is useful beyond correctness. It supports inspection, supervision, and future training (e.g., trace-level preference learning or step-level distillation), aligning with established views of learning-through-interaction (Chi and Wylie, 2014; Kapur, 2008).

In this paper, we propose **CPA** (**C**ognitive **P**lanning **A**gent). CPA is not a new trained model. It is a runtime agent/controller that wraps a backbone LLM at inference time. CPA follows a standardized four-stage pipeline: (i) **task deep decomposition** into a panoramic state, (ii) **plan template synthesis** via a generate–simulate–evaluate–revise loop, (iii) **step-wise execution** to produce a structured planning trace, and (iv) **output derivation** from the deliverable step in the trace. This structure reduces the model's degrees of freedom. Instead of freely deciding "what to do next" at each token, the model executes within an explicit plan, complementing prior planning paradigms such as plan-then-execute (Wang et al., 2023) and skeleton-based prompting (Ning et al., 2023).

We evaluate CPA on MATHTUTORBENCH (Macina et al., 2025). The benchmark includes seven tasks that cover core tutoring abilities: problem solving, Socratic questioning, solution correctness checking, mistake localization, mistake correction, and two pedagogical response tasks. Across these tasks, CPA improves overall tutoring quality. It also produces more consistent intermediate traces, which can support analysis and supervision.

Contributions. (1) We formalize tutoring and related abilities as *complex cognitive planning tasks* that benefit from stable planning and interpretable intermediate traces, consistent with classic tutoring and learning theories (Vygotsky, 1978; Chi and Wylie, 2014; Hattie and Timperley, 2007). (2) We propose CPA, a training-free runtime planning architecture whose key component is *instance-specific plan template synthesis* via generate–simulate–evaluate–revise. (3) On MATHTUTORBENCH (Macina et al., 2025) (seven tutoring-related tasks), CPA achieves strong gains, and ablations isolate the role of each module.

2 Related Work

2.1 Planning for complex tasks

Complex planning typically involves goals, constraints, structured steps, and deliverables. LLM-based planning via chain-of-thought and its variants (Wei et al., 2022; Kojima et al., 2022) can improve multi-step reasoning, while further techniques decompose or stabilize reasoning trajectories, such as least-to-most prompting (Zhou et al., 2022), self-consistency (Wang et al., 2022), and decomposed prompting (Khot et al., 2022). Beyond prompting, agentic frameworks incorporate explicit action/interaction loops, e.g., ReAct (Yao et al., 2022), Tree-of-Thoughts (Yao et al., 2023), and iterative self-improvement via self-feedback (Madaan et al., 2023) or verbal reinforcement learning signals (Shinn et al., 2023). However, despite these advances, LLM reasoning can remain unstable and may not reliably self-correct without structured control (Huang et al., 2023).

2.2 LLM-based tutoring dialogue

Tutoring aims to guide learners through structured reasoning rather than only providing answers. Classic educational theory emphasizes guided participation and scaffolding (Vygotsky, 1978; Anghileri, 2006), feedback (Hattie and Timperley, 2007), and engagement modes (Chi and Wylie, 2014), and empirical meta-analyses compare human tutoring with intelligent tutoring systems (VanLehn, 2011). Earlier systems such as Cognitive Tutors (Anderson et al., 1995) and AutoTutor (Graesser et al., 2012) operationalize structured pedagogical strategies. Recent LLM-based tutors aim to elicit Socratic behaviors and improve learning outcomes via prompting or model adaptation, including Socratic prompting (Chang, 2023), LearnLM (Team et al., 2024), SocraticLM (Liu et al., 2024), and science tutoring settings (Chevalier et al., 2024). Multi-turn pedagogical control strategies (e.g., hierarchical questioning and planning) are also explored in related interactive settings (Kargupta et al., 2024). Nevertheless, end-to-end generation often mixes understanding, planning, and execution in one pass, making intermediate intentions hard to supervise. CPA instead focuses on a runtime planning structure that makes task understanding, plan synthesis, and step-wise execution explicit.

2.3 Plan-then-execute prompting

Plan-and-solve prompting (Wang et al., 2023) and skeleton-based prompting (Ning et al., 2023) separate planning from execution, improving controllability and structure. Our work shares this motivation, but differs in two ways. First, we extract a panoramic task state to stabilize instance understanding. Second, we treat planning as *template synthesis* solved by an explicit generate–simulate–evaluate–revise loop, which targets instance-level executability and trace consistency.

2.4 Math reasoning and tutoring benchmarks

Math reasoning benchmarks and training setups have been widely studied, including GSM-style verifiers (Cobbe et al., 2021), the MATH dataset (Hendrycks et al., 2021), and large-scale quantitative reasoning with LMs (Lewkowycz et al., 2022). Math-specific instruction data and models further push capabilities (Luo et al., 2023; Yu et al., 2023; Azerbayev et al., 2023). For pedagogical dialogue, datasets such as MathDial (Macina et al., 2023) provide tutoring-oriented interactions, and MATHTUTORBENCH (Macina et al., 2025) benchmarks open-ended pedagogical capabilities of LLM tutors.

3 CPA: A Metacognition-Driven Cognitive Planning Agent for Pedagogical Dialogue

3.1 Task Definition: Cognitive Planning in Tutoring Dialogue

We study tutoring dialogue tasks where the system receives a problem statement, a student's partial solution: dialogue history, and task instructions. Each turn requires producing a deliverable (e.g., an answer, a verdict, an error index, or a guidance move) that satisfies task requirements. Some tasks include explicit constraints (e.g., output format or "no direct answer"), but our focus is broader: we aim to make the planning process stable and the intermediate trace interpretable, consistent with scaffolding-oriented tutoring principles (Anghileri, 2006; Hattie and Timperley, 2007).

3.2 Overview of CPA

At turn t, CPA follows four stages: **M1 Task Deep Decomposition** $\rightarrow$ **M2 Plan Template Synthesis** $\rightarrow$ **M3 Step-wise Execution** $\rightarrow$ **M4 Output Derivation**. Formally:

$$(A_t, E_t, \hat{p}_t) = f_{\text{perc}}(\mathcal{H}_{\leq t}, p_t; \mathcal{M}_{t-1}), \quad (1)$$

$$S_t = f_{\text{temp}}(A_t, \hat{p}_t), \quad (2)$$

$$(C_t, \Delta_t) = f_{\text{exec}}(S_t, A_t, \hat{p}_t), \quad (3)$$

$$y_t = f_{\text{out}}(C_t, S_t, A_t), \quad (4)$$

where $\mathcal{H}_{\leq t}$ is dialogue history, p_t is the current input, A_t is the panoramic task state, and $\hat{p}_t$ is a condensed instance summary. S_t is the synthesized plan template (skeleton), C_t is the step-wise execution trace, and y_t is the final response. E_t optionally stores short evidence spans aligned to key state fields, which can support analysis. Δ_t is a lightweight monitoring signal used to detect execution drift.

3.3 Multi-turn State

CPA maintains a persistent memory $\mathcal{M}_t = \langle A_t, S_t, k_t, \tau_t, \text{summary}(C_t) \rangle$, where k_t is a step pointer and τ_t is the deliverable step. On new turns, CPA updates the state incrementally and re-synthesizes the template when the goal, bottleneck, or requirements change.

3.4 M1: Task Deep Decomposition

M1 maps $(\mathcal{H}_{\leq t}, p_t)$ to a panoramic task state:

$$A_t = \langle A_t^{\text{ctx}}, A_t^{\text{goal}}, A_t^{\text{bar}}, A_t^{\text{cst}} \rangle, \quad (5)$$

capturing context, the turn objective and success criteria, the main bottlenecks (e.g., where the student is stuck), and task requirements/constraints. M1 also constructs $\hat{p}_t$ as a condensed instance summary. This summary reduces long-context noise and provides a stable conditioning signal for M2 and M3. In our implementation, $\hat{p}_t$ is produced by one LLM call that rewrites the instance into a compact, structured form (problem, student state, required deliverable, key constraints). The optional evidence E_t can be extracted in the same call by selecting short spans from the input that justify key fields (e.g., the required deliverable or the student's claimed step).

3.5 M2: Plan Template Synthesis via Generate–Simulate–Evaluate–Revise

M2 is the core of CPA. It synthesizes an instance-specific plan template S_t. A template specifies: (i) an ordered list of step roles, (ii) step-local requirements derived from A_t, and (iii) a deliverable pointer τ_t indicating which step will produce the

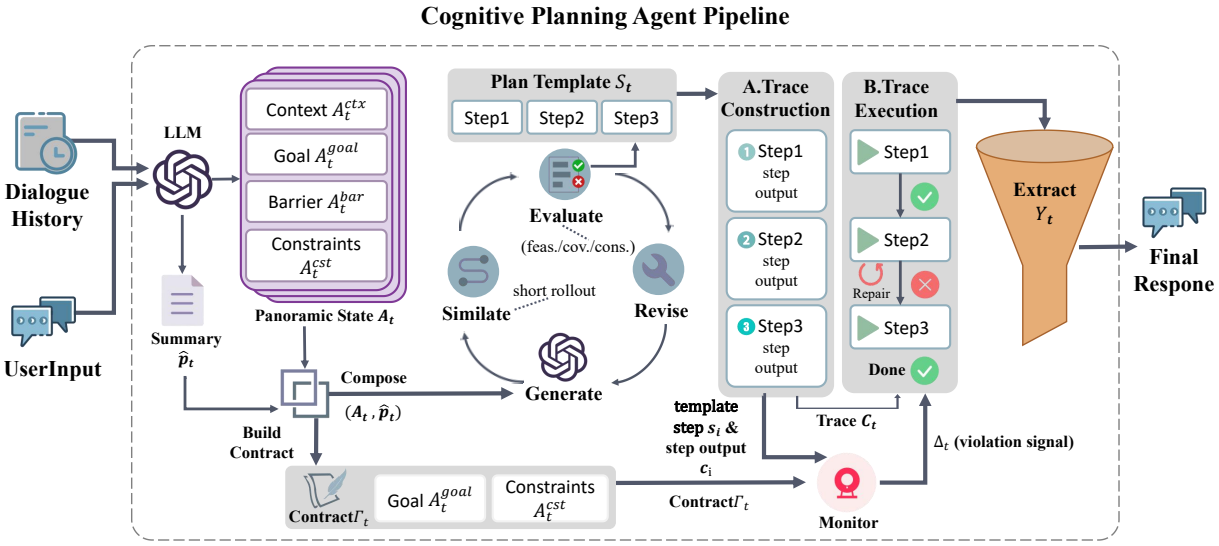


Figure 2: **CPA overview.** CPA (M1) builds a panoramic task state, (M2) synthesizes an instance-specific plan template via generate–simulate–evaluate–revise, (M3) executes the template step-by-step to produce a structured planning trace, and (M4) derives the final response from the deliverable step. During execution, a lightweight monitor checks step outputs against a compiled *contract* Γ_t (goal + constraints) and emits a drift/violation signal Δ_t for repair or re-synthesis.

Algorithm 1 M2: Plan Template Synthesis (Generate–Simulate–Evaluate–Revise)

Require: Panoramic state A_t, condensed summary $\hat{p}_t$
Ensure: Plan template S_t with deliverable step τ_t

$S_t^{(0)} \leftarrow \textsc{Generate}(A_t, \hat{p}_t)$
for $j = 0, 1, \ldots, J-1$ **do**
  $\tilde{C}_t^{(j)} \leftarrow \textsc{Simulate}(S_t^{(j)}, A_t, \hat{p}_t)$
  $\Delta^{(j)} \leftarrow \textsc{Evaluate}(\tilde{C}_t^{(j)}, A_t)$
  if $\Delta^{(j)} = \textsc{Pass}$ **then**
    return $S_t^{(j)}$
  else
    $S_t^{(j+1)} \leftarrow \textsc{Revise}(S_t^{(j)}, \Delta^{(j)}, A_t)$
  end if
end for
return $S_t^{(J)}$

external artifact. The key design is that S_t is *not fixed*. It is synthesized for the current instance via a closed-loop procedure: **generate** candidates, **simulate** a short dry-run, **evaluate** template quality, and **revise** the template.

Generate. CPA first generates a small set of candidate templates from $(A_t, \hat{p}_t)$. Each candidate is a concise skeleton with step roles such as *understand*, *verify*, *locate error*, *derive next action*, and *compose guidance*. The output is structured (a list of steps with role tags and brief step goals).

Simulate (short dry-run). For each candidate template $S_t^{(j)}$, CPA performs a short look-ahead rollout to produce a shadow trace $\tilde{C}_t^{(j)}$. This is not a deep proof search. It is a lightweight "dry-run" that checks whether the template is executable and whether it tends to produce the intended deliverable. The simulation uses $\hat{p}_t$ to keep the rollout short and stable.

Evaluate. CPA evaluates each candidate using a small set of general criteria: (1) **feasibility**: can the steps be executed with available information? (2) **coverage**: do the steps cover the goal (and key bottlenecks) without skipping necessary checks? (3) **consistency**: does the simulated deliverable match the required output type? (4) **economy**: is the granularity appropriate (no redundant steps)? When tasks include explicit constraints (e.g., strict format), we add simple deterministic checks during evaluation. The evaluation can be implemented by a lightweight verifier prompt (LLM-as-judge; Zheng et al., 2023; Liu et al., 2023) plus optional rule checks for formats.

Revise. If a candidate fails, CPA revises it minimally. Typical revisions include adjusting step order, adding a missing verification step, or changing the deliverable step. The loop repeats until a satisfactory template is found or a maximum number of iterations is reached.

3.6 M3: Step-wise Execution to Produce a Planning Trace

Given $S_t = \{s_i\}_{i=1}^{T}$, CPA executes steps sequentially:

$$(C_t, \Delta_t) = f_{\text{exec}}(S_t, A_t, \hat{p}_t), \quad C_t = \{c_i\}_{i=1}^{T}. \tag{6}$$

Each step produces a concise *intermediate result* (not exposed to the user). The trace is structured by the template roles. This makes the process more stable and easier to inspect than free-form generation.

Contract and monitoring. To reduce drift and enforce task requirements, CPA compiles a lightweight *contract*

$$\Gamma_t = \langle A_t^{\text{goal}}, A_t^{\text{cst}} \rangle, \tag{7}$$

which captures the turn objective (deliverable type / success criteria) and explicit constraints (e.g., strict format or "no direct answer"). During step-wise execution, a monitor checks each step output c_i (and its step role s_i) against Γ_t, producing a drift/violation signal Δ_t. If drift persists (e.g., missing deliverable type, skipping required checks, violating constraints), CPA triggers local repair or returns to M2 to re-synthesize the template.

3.7 M4: Output Derivation from the Deliverable Step

M4 aligns to the deliverable step τ_t and derives the final response:

$$y_t = f_{\text{out}}(C_t, S_t, A_t). \tag{8}$$

For structured deliverables, M4 formats the output deterministically when possible. For natural-language guidance, M4 performs minimal rewriting to fit the dialogue context, while keeping the deliverable content unchanged. This reduces the risk that the model reinterprets the goal at the final stage.

3.8 Summary

CPA organizes complex cognitive planning into four stages and produces an interpretable planning trace (Figure 3). M1 builds a panoramic understanding of the instance and a condensed summary; M2 synthesizes an executable plan template via generate–simulate–evaluate–revise; M3 executes the template step by step to produce a structured trace with contract-based monitoring; and M4 derives the final output from the deliverable step. This design supports stable planning and makes the intermediate process available for inspection and supervision.

Task	Deliverable	Key constraints
Problem solving	Final answer	Standard format (e.g., boxed)
Socratic questioning	Next question	No answer leakage; single question
Solution checking	Verdict + Evidence	Strict label (Correct/Incorrect); span extraction
Mistake localization	Error span / Step idx	Exact step index or text span
Mistake correction	Corrected step	Coherence with previous steps
Scaffolding response	Tutoring response	No direct answer; pedagogical guidance
Instruction-following	Tutoring response	Strict adherence to specific instructions

Table 1: **Task deliverables and contracted constraints.** CPA compiles goal + constraints into Γ_t and checks step outputs during execution.

4 Experiments

We evaluate CPA on MATHTUTORBENCH (Macina et al., 2025). The benchmark contains seven tutoring-related tasks that stress different aspects of cognitive planning: problem solving, Socratic questioning, solution correctness checking, mistake localization, mistake correction, scaffolding response generation, and instruction-following tutoring responses. We follow the benchmark's official evaluation protocol.

4.1 Tasks and Contracted Constraints

MATHTUTORBENCH contains heterogeneous tutoring tasks with different deliverables and explicit constraints. CPA compiles task-specific requirements into the execution contract Γ_t (Section 3) and monitors step outputs for drift/violations. Table 1 summarizes the deliverables and typical constraint types.

4.2 Benchmark and Metrics

We report the benchmark metrics for each task: problem solving accuracy, Socratic questioning BLEU (Papineni et al., 2002), solution correctness (F1), mistake localization (mF1), mistake correction accuracy, and pairwise win rates for the two pedagogical response tasks (with a strict/hard subset). For open-ended responses, the benchmark relies on LLM-based preference judgments (LLM-as-a-judge; Zheng et al., 2023; Liu et al., 2023).

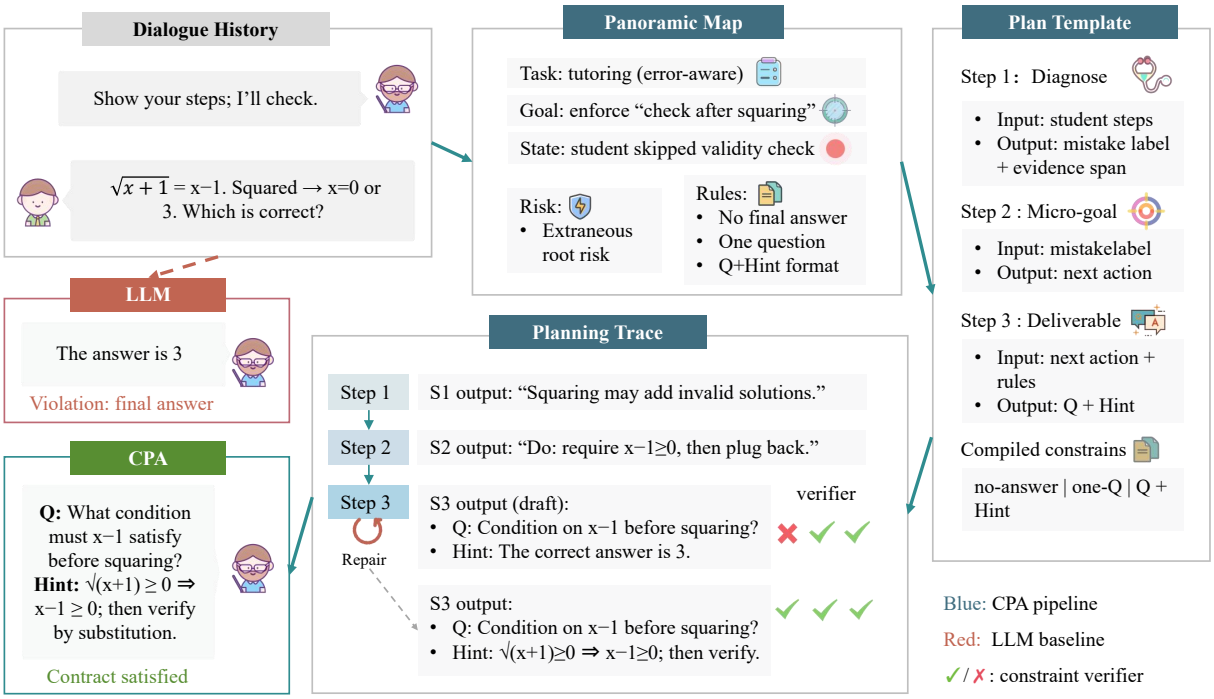


Figure 3: **What CPA produces internally.** Example intermediate artifacts for one tutoring instance: panoramic state, plan template, step-wise planning trace, and contract-based checking/repair signals. This makes the tutoring process inspectable beyond the final response.

4.3 Compared Methods

We evaluate CPA against strong prompting baselines, specialized tutoring models, and a state-of-the-art proprietary model.

Prompting baselines (controlled backbone). We compare CPA with three representative planning/prompting methods: **CoT** (Wei et al., 2022), **Plan-and-Solve** (Wang et al., 2023), and **Skeleton-of-Thought** (Ning et al., 2023). All prompting baselines are instantiated on the same backbone model (QWEN2.5-7B-INSTRUCT) for controlled comparison.

Benchmark fine-tuned tutoring models. We include benchmark-reported specialized tutoring models: **LearnLM-1.5-Pro** (Team et al., 2024), **Llemma-7B-ScienceTutor** (Chevalier et al., 2024), and **Qwen2.5-7B-SocraticLM** (Liu et al., 2024).

Backbone models. We use QWEN2.5-7B-INSTRUCT (Bai et al., 2023) as our primary open-source backbone. We also include **GPT-4o** (Achiam et al., 2023) as a strong proprietary baseline to assess CPA's scalability to stronger models.

4.4 Implementation Details and Reproducibility

We implement CPA as a runtime controller around a backbone LLM. Each turn runs: (M1) a call that extracts the panoramic state A_t and a condensed summary $\hat{p}_t$; (M2) a bounded template-synthesis loop that generates candidate templates, performs short dry-run simulations, evaluates template quality, and revises if needed; (M3) step-wise execution following the selected template; and (M4) output derivation from the deliverable step. During (M3), a monitor checks each step output against the contract $\Gamma_t = \langle A_t^{\text{goal}}, A_t^{\text{cst}} \rangle$ and triggers local repair or template re-synthesis when violations persist.

To support reproducibility and fair comparison, we explicitly report: (i) decoding settings: temperature $T = 0.7$, top-$p = 0.95$, and max new tokens set to 1024; (ii) CPA loop limits: $K = 3$ candidate templates, $J = 3$ maximum revise iterations, and simulation rollout length restricted to 3 steps; (iii) monitoring/repair budgets: maximum 2 local repair attempts before fallback to re-synthesis; and (iv) randomness control: all results are averaged over 3 runs with fixed seeds (42, 43, 44).

4.5 Main Results

Table 2 reports results across all tasks. CPA improves overall tutoring quality and remains com-

petitive on core math competence. Gains are especially clear on tasks that require stable planning across diagnosis, correction, and guidance, which aligns with CPA's goal of producing a consistent planning trace and controllable pedagogical interaction (Anghileri, 2006; Hattie and Timperley, 2007).

4.6 Constraint Violations and Trace Stability

Beyond end-task scores, we analyze controllability signals that are directly tied to CPA's design. In strict tutoring settings, models may drift by revealing final answers, asking multiple questions at once, skipping required checks, or violating strict formats. The benchmark reports a constraint violation rate (**BVR**) on strict/hard subsets. CPA is designed to reduce such violations via contract-based monitoring and repair.

Violation rate (BVR). We report BVR to directly quantify controllability gains. On the strict/hard subsets of MATHTUTORBENCH, baseline models often exhibit BVRs exceeding 15%, whereas CPA achieves a low BVR of 0.06 (Table 3), indicating strong adherence to pedagogical constraints.

Monitoring/repair frequency. The monitor outputs a drift/violation signal Δ_t during step-wise execution. Analysis of logged traces shows that monitoring triggers local repair in approximately 18% of turns, preventing potential violations from propagating to the final response.

4.7 Ablation Study

We ablate modules to isolate which components contribute to overall tutoring quality and process stability. We report strict/hard win rates and a constraint violation rate (**BVR**), computed as the fraction of responses flagged by the benchmark checker as violating strict formats or explicit "no direct answer" constraints.

4.8 Qualitative Analysis

Figure 3 illustrates the intermediate artifacts produced by CPA, including the panoramic state, plan template, step-wise trace, and contract-based checking signals. This intermediate trace makes tutoring behavior inspectable beyond the final response and helps diagnose failure modes.

In our analysis, common baseline failures on strict/hard subsets include: (i) *answer leakage* (revealing the final answer despite "no direct answer"), (ii) *format drift* (violating one-question or required output schema), and (iii) *cognitive drift* (skipping verification steps such as validity checks in algebraic manipulation). CPA mitigates these errors by selecting explicit micro-goals in the synthesized template and enforcing Γ_t during execution, with repair/re-synthesis triggered by Δ_t when violations persist.

4.9 Discussion

M2 improves instance-level plan quality. Removing the M2 synthesis loop degrades strict/hard win rates and increases violations. This suggests that template quality is a bottleneck. The dry-run simulation helps by filtering templates that skip necessary steps or produce the wrong deliverable type.

Monitoring and output derivation improve trace stability. Disabling M3 monitoring increases violations, which indicates that drift can emerge during step-wise execution even with a reasonable template. Disabling M4 increases errors at the final stage, which supports the need to derive outputs from the deliverable step instead of re-planning in the last pass.

5 Limitations and Ethics

CPA is a runtime planning agent and can add inference overhead due to template synthesis and step-wise execution. While the planning trace improves interpretability, it does not guarantee factual correctness. In deployment, CPA should be paired with appropriate safety policies for educational settings and transparent communication that the system is assistive rather than authoritative. More broadly, tutoring systems should follow established best practices around feedback and learner support (Hattie and Timperley, 2007; VanLehn, 2011).

6 Conclusion

We presented CPA, a metacognition-driven runtime planner for complex cognitive planning tasks, using tutoring dialogue as a case study. CPA extracts a panoramic task state, synthesizes an instance-specific plan template via generate–simulate–evaluate–revise, executes the template to produce a structured planning trace with contract-based monitoring, and derives the final output from the deliverable step. On MATHTUTORBENCH (Macina et al., 2025), CPA improves overall tutoring quality and produces more consistent intermediate traces. These results suggest that metacognition-inspired template synthesis and trace-based execu-

Model / Method	Math		Understanding			Tutoring Win ↑			
	Prob.	Socr.	Corr.	Loc.	Fix.	Scaff.	Inst.	Scaff.†	Inst.†
Prompting Baselines (Backbone: Qwen2.5-7B-Instruct)									
CoT	0.74	0.27	0.64	0.31	0.11	0.54	0.60	0.38	0.41
Plan-and-Solve	0.75	0.28	0.65	0.33	0.12	0.56	0.62	0.40	0.43
Skeleton-of-Thought	0.73	0.28	0.64	0.32	0.11	0.55	0.61	0.39	0.42
Fine-tuned Tutors									
LearnLM-1.5-Pro	**0.94**	0.32	**0.75**	**0.57**	0.74	0.64	0.68	0.66	0.67
Llemma-7B-ScienceTutor	0.62	0.29	0.66	0.29	0.16	0.37	0.48	0.38	0.42
Qwen2.5-7B-SocraticLM	0.73	0.32	0.05	0.39	0.23	0.39	0.39	0.28	0.28
Ours & GPT-4o									
CPA-7B (Ours)	0.76	0.30	0.68	0.44	0.52	0.69	0.71	0.76	0.78
GPT-4o	0.90	**0.48**	0.67	0.37	**0.84**	0.50	0.82	0.46	0.70
CPA-GPT-4o (Ours)	0.92	0.44	0.74	0.55	**0.84**	**0.92**	**0.92**	**0.94**	**0.88**

Table 2: **Main results on MATHTUTORBENCH (Macina et al., 2025). Prob.**: problem solving accuracy; **Socr.**: Socratic questioning BLEU (Papineni et al., 2002); **Corr.**: solution correctness (F1); **Loc.**: mistake localization (mF1); **Fix**: mistake correction accuracy; **Scaff.**: scaffolding win rate; **Inst.**: tutoring instruction-following win rate. † denotes the strict/hard subset. We compare prompting strategies on QWEN2.5-7B-INSTRUCT and evaluate performance scaling on **GPT-4o**.

Variant (CPA-7B)	Scaff.†	Inst.†	BVR↓
Full CPA	0.76	0.78	0.06
w/o M1 (no state)	0.61	0.63	0.15
w/o M2-loop (1-shot template)	0.66	0.67	0.12
w/o M2-sim (no dry-run)	0.69	0.70	0.10
w/o M3-monitor (no drift chk)	0.62	0.64	0.17
w/o M4 (direct output)	0.65	0.66	0.14

Table 3: **Ablation on CPA-7B** (†: strict/hard subset). BVR denotes the constraint violation rate measured by the benchmark checker (lower is better). We remove one component at a time: M1 panoramic state, the M2 synthesis loop, M2 dry-run simulation, M3 drift monitoring, and M4 output derivation.

tion are a practical path toward more stable and inspectable LLM planning.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. 1995. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207.

Julia Anghileri. 2006. Scaffolding practices that enhance mathematics learning. *Journal of Mathematics Teacher Education*, 9(1):33–52.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Edward Y Chang. 2023. Prompting large language models with the socratic method. In *2023 IEEE 13th annual computing and communication workshop and conference (CCWC)*, pages 0351–0360. IEEE.

Alexis Chevalier, Jiayi Geng, Alexander Wettig, Howard Chen, Sebastian Mizera, Toni Annala, Max Jameson Aragon, Arturo Rodríguez Fanlo, Simon Frieder, Simon Machado, and 1 others. 2024. Language models as science tutors. *arXiv preprint arXiv:2402.11111*.

Michelene TH Chi and Ruth Wylie. 2014. The icap framework: Linking cognitive engagement to active learning outcomes. *Educational psychologist*, 49(4):219–243.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Arthur C Graesser, Sidney D'Mello, Xiangen Hu, Zhiqiang Cai, Andrew Olney, and Brent Morgan.

2012. Autotutor. In *Applied natural language processing: Identification, investigation and resolution*, pages 169–187. IGI Global Scientific Publishing.

John Hattie and Helen Timperley. 2007. The power of feedback. *Review of educational research*, 77(1):81–112.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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. *arXiv preprint arXiv:2310.01798*.

Manu Kapur. 2008. Productive failure. *Cognition and instruction*, 26(3):379–424.

Priyanka Kargupta, Ishika Agarwal, Dilek Hakkani-Tur, and Jiawei Han. 2024. Instruct, not assist: Llm-based multi-turn planning and hierarchical questioning for socratic code debugging. *arXiv preprint arXiv:2406.11709*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.

Jiayu Liu, Zhenya Huang, Tong Xiao, Jing Sha, Jinze Wu, Qi Liu, Shijin Wang, and Enhong Chen. 2024. Socraticlm: Exploring socratic personalized teaching with large language models. *Advances in Neural Information Processing Systems*, 37:85693–85721.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Jakub Macina, Nico Daheim, Sankalan Chowdhury, Tanmay Sinha, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. 2023. Mathdial: A dialogue tutoring dataset with rich pedagogical properties grounded in math reasoning problems. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5602–5621.

Jakub Macina, Nico Daheim, Ido Hakimi, Manu Kapur, Iryna Gurevych, and Mrinmaya Sachan. 2025. Mathtutorbench: A benchmark for measuring open-ended pedagogical capabilities of llm tutors. *arXiv preprint arXiv:2502.18940*.

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.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2023. Skeleton-of-thought: Large language models can do parallel decoding. *Proceedings ENLSP-III*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

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.

LearnLM Team, Abhinit Modi, Aditya Srikanth Veerubhotla, Aliya Rysbek, Andrea Huber, Brett Wiltshire, Brian Veprek, Daniel Gillick, Daniel Kasenberg, Derek Ahmed, and 1 others. 2024. Learnlm: Improving gemini for learning. *arXiv preprint arXiv:2412.16429*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Kurt VanLehn. 2011. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational psychologist*, 46(4):197–221.

Lev S Vygotsky. 1978. *Mind in society: The development of higher psychological processes*, volume 86. Harvard university press.

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. *arXiv preprint arXiv:2305.04091*.

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

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.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.