

TAC-CoT: Tool-Empowered Implicit Chain-of-Thought via Interleaving Explicit Tokens

Juseon Do¹ Sangmin Bae² Jeesu Jung^{1,*} Hwanjun Song^{1,3,*}

¹Data Intelligence System Lab, KAIST

²Optimization and Statistical Inference Lab, KAIST AI

³Cluvion

dojuseon@gmail.com {bsmn0223, jeesu_jung, songhwanjun}@kaist.ac.kr

Abstract

LLM-based agentic systems are increasingly combined with CoT reasoning to solve tasks that require accurate reasoning, external knowledge, and multi-step deliberation. Yet, explicit CoT incurs substantial generation overhead, while implicit CoT, which replaces textual rationales with compact continuous tokens, does not naturally support the explicit communication interface required for agentic tool-use. We propose TAC-CoT, a tool-empowered adaptive implicit CoT framework that replaces explicit reasoning traces with compact latent reasoning segments while retaining textual tokens only for tool calls, observations, and final answers. This interleaved design enables agentic tool-use without verbalizing every intermediate reasoning step. Experiments on agentic math and QA benchmarks show that TAC-CoT improves accuracy by up to 8.8 points while reducing model forward passes by up to $6.8\times$. Furthermore, analyses show that TAC-CoT improves tool-use quality in both single- and multi-tool settings.

1 Introduction

To enhance the problem-solving ability of large language models (LLMs), a wide range of reasoning-oriented methods have been explored. In particular, for tasks that require accurate and reliable reasoning, such as mathematical reasoning and knowledge-intensive question answering, research has evolved toward Chain-of-Thought (CoT) reasoning, which improves models’ internal reasoning capabilities through intermediate reasoning steps (Wei et al., 2022; DeepSeek-AI et al., 2025b), as well as agentic tool-use frameworks, which enhance rigor and factuality by enabling models to interact with external tools during problem solving (Zhang et al., 2026; Kim et al., 2026).

*Corresponding authors.

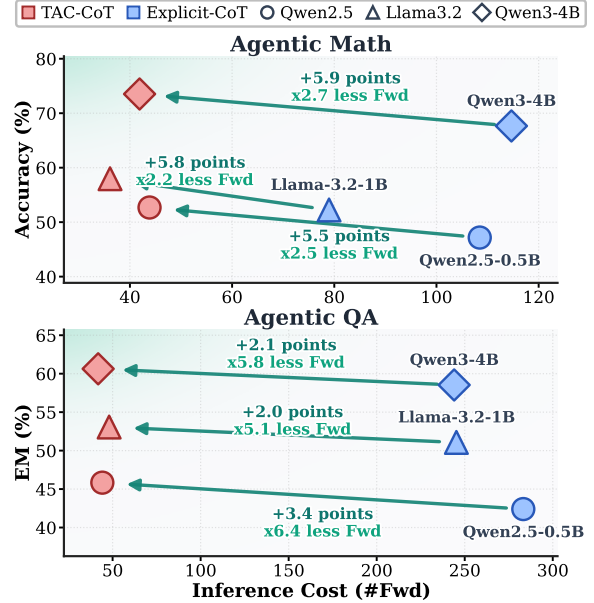


Figure 1: TAC-CoT improves over the Explicit-CoT baseline in both agentic math and question-answering (QA) settings. The two methods use the same tool-using agents and differ only in their reasoning backbone: TAC-CoT uses implicit CoT, whereas the baseline uses explicit CoT. #Fwd denotes the number of model forward passes (Kuzina et al., 2026; Wei et al., 2026).

Most prior work adopts the explicit CoT paradigm, where intermediate reasoning is represented as human-readable natural-language tokens. This paradigm is attractive because it is directly compatible with autoregressive generation and provides an interpretable interface between reasoning and action (Kojima et al., 2022; Yao et al., 2023). However, verbalizing the entire reasoning process substantially increases inference cost as reasoning traces grow longer and makes the model more susceptible to error accumulation across intermediate steps (Han et al., 2025; Wu et al., 2026).

Implicit CoT has emerged as an alternative paradigm that replaces readable natural-language rationales with compact *continuous thought tokens*. By performing reasoning in non-verbal latent spaces, implicit CoT aims to retain the benefits

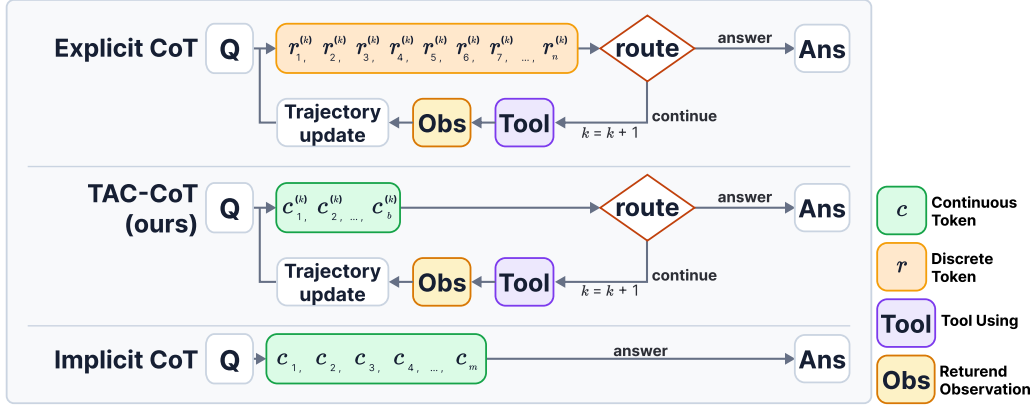


Figure 2: Overview of TAC-CoT compared with the Explicit-CoT agent and implicit-CoT methods. TAC-CoT uses implicit CoT as its reasoning backbone while preserving interaction segments in explicit natural language. This design enables implicit CoT to use tools and allocate query-dependent implicit reasoning budgets, improving efficiency, performance, and robustness over the Explicit-CoT agent. Here, n denotes the number of discrete tokens generated by the Explicit-CoT agent at each step, b denotes the number of continuous tokens in each implicit reasoning segment of TAC-CoT, and m denotes the fixed continuous-token budget used by prior implicit-CoT methods.

of multi-step reasoning while mitigating the inference cost and vulnerability to error accumulation of lengthy textual traces (Shen et al., 2025).

Despite these benefits, implicit CoT methods are ill-suited for LLM-based agent systems. They rely on a fixed number of implicit tokens to carry the entire reasoning process of every query, which gives rise to two limitations. First, since the implicit-token budget is consumed in a single uninterrupted block before the final answer (Deng et al., 2024; Wei et al., 2026), there is no intermediate point at which the model can emit an explicit tool call or ingest an observation, so tool use is structurally infeasible. Second, the same token count is applied to every input regardless of its difficulty or the number of intermediate decision points it requires (Wu et al., 2026; Aggarwal et al., 2026). As a result, prior implicit CoT methods cannot interact with external environments and under-allocate reasoning capacity to more difficult queries.

In this paper, we propose TAC-CoT (Tool-empowered Adaptive Continuous CoT), which *interleaves compact implicit reasoning segments with explicit tokens for tool calls, observations, and final answers*. Rather than carrying the entire reasoning process in a single block of implicit tokens, TAC-CoT reasons within a short implicit segment and then either invokes a tool or produces the answer, resuming with another segment after each tool observation. This interleaving simultaneously addresses both limitations of prior implicit CoT, as explicit tokens between segments make tool use executable, while the number of segments, determined on the fly, makes the reasoning budget

query-adaptive. To train this behavior, we automatically construct tool-use trajectories from reasoning datasets and supervise the model to perform compact latent reasoning between tool invocations.

Our contributions are summarized as follows.

- (1) We extend implicit CoT to tool-empowered agentic systems, enabling compact continuous reasoning in iterative reasoning–action workflows.
- (2) We show that TAC-CoT improves reasoning performance while substantially reducing the number of reasoning tokens compared with explicit CoT-style reasoning by interleaving continuous thought tokens with explicit tokens.
- (3) We introduce an adaptive token-allocation mechanism that flexibly adjusts the length of implicit reasoning according to query difficulty.
- (4) As highlighted in Figure 1, TAC-CoT improves downstream task performance while using the forward reasoning budget more efficiently. Our analysis shows that TAC-CoT benefits from query-adaptive implicit reasoning budgets, improves tool-using scenarios, and mitigates error accumulation through implicit reasoning.

2 TAC-CoT: Tool-Empowered Implicit CoT via Interleaving Explicit Tokens

We explain TAC-CoT, a tool-empowered implicit CoT framework that interleaves latent reasoning segments with explicit, executable tool interactions. We first describe TAC-CoT’s inference procedure (§2.1), then how we construct training trajectories (§2.2), and finally the learning objective (§2.3). Algorithm 1 shows its inference procedure.

2.1 Tool-Interleaved Implicit CoT with Query-Adaptive Budgets

The key idea of TAC-CoT is to jointly leverage implicit reasoning tokens and textual tokens for tool-use trajectories. In this process, the length of the reasoning tokens is adaptively adjusted according to the query. As shown in Figure 2, this produces an interleaved trajectory that preserves the executable interface of tool-using agents without verbalizing every intermediate step.

Given a query q , we consider an *explicit* tool-using CoT trajectory as:

$$\tau_E = (\mathcal{R}^{(1)}, u^{(1)}, o^{(1)}, \dots, \mathcal{R}^{(k)}, a), \quad (1)$$

where $\mathcal{R}^{(i)} = (r_1^{(i)}, \dots, r_n^{(i)})$ is the i -th explicit reasoning segment of n discrete tokens; $u^{(i)}$ and $o^{(i)}$ are the corresponding tool call and observation, and a is the final answer. The last segment $\mathcal{R}^{(k)}$ produces no tool call, as the model uses it to consolidate prior observations before generating a . TAC-CoT constructs the corresponding *interleaving* trajectory τ_I by replacing each explicit reasoning segment $\mathcal{R}^{(i)}$ with an implicit segment $C^{(i)} = (c_1^{(i)}, \dots, c_b^{(i)})$ of b ($< n$) continuous thought tokens, while keeping $u^{(i)}$, $o^{(i)}$, and a in their original explicit textual form:

$$\tau_I = (C^{(1)}, u^{(1)}, o^{(1)}, \dots, C^{(k)}, a). \quad (2)$$

Each continuous thought token $c_t^{(i)}$ is obtained by applying a projection layer to the last-layer hidden state at the current position and is fed back to the model via input embeddings, bypassing the vocabulary projection; following CODI (Shen et al., 2025), this projection layer separates the latent thought space from the token embedding space.

Deciding When to Call a Tool. A key design decision of TAC-CoT is *when* to interrupt latent reasoning for an external interaction. At the end of every implicit segment $C^{(i)}$, TAC-CoT compares the logits of the tool call $u^{(i)}$ and answer options to decide the next step. If the tool-call logit is higher, TAC-CoT emits a tool call, appends the returned observation $o^{(i)}$ to the context, and starts another implicit segment. Otherwise, the current segment becomes the consolidating segment $C^{(k)}$, from which the final answer is generated. Unlike prior implicit CoT methods that consume a fixed implicit budget before any explicit output, this per-segment routing is what makes tool use feasible under implicit reasoning.

Algorithm 1: TAC-CoT inference

Input: Query q , model f_θ , tool executor $\mathcal{E}$, implicit segment length b

Output: Final answer a

```

1  $\tau \leftarrow []$  // prefix of  $\tau_I$ 
2 repeat
    // Implicit reasoning segment
3   for  $t \leftarrow 1$  to  $b$  do
4      $h_t \leftarrow \text{LastHidden}_\theta(q \circ \tau)$ 
    // Continuous thought token
5      $c_t \leftarrow \text{Proj}_\theta(h_t)$ 
6      $\tau \leftarrow \tau \circ [c_t]$ 
    // Route selection
7    $(\ell_{\text{act}}, \ell_{\text{ans}}) \leftarrow \text{RouteLogits}_\theta(q \circ \tau)$ 
8   if  $\ell_{\text{act}} > \ell_{\text{ans}}$  then
9      $u \leftarrow \text{DecodeTool}_\theta(q \circ \tau)$ 
10     $o \leftarrow \mathcal{E}(u)$ 
11     $\tau \leftarrow \tau \circ [u, o]$ 
12 until  $\ell_{\text{ans}} \geq \ell_{\text{act}}$ 
13  $a \leftarrow \text{DecodeAns}_\theta(q \circ \tau)$ 
14 return  $a$ 

```

Query-Adaptive Continuous-Thought Token Budget.

Since the tool call decision is made after every implicit segment, the total reasoning budget becomes *query-adaptive*. Simple queries may terminate after a single segment, while more complex queries can continue through multiple tool interactions. Thus, b is fixed only locally within each segment. In contrast, if TAC-CoT terminates after k implicit segments $C^{(i)}$, it uses kb continuous tokens in total, where k is determined dynamically during inference.

2.2 Dataset Construction for Agentic CoT

Realizing our interleaved inference mechanism in Eq (2) requires training data that exposes the explicit counterpart τ_E of the interleaved trajectory τ_I . That is, each example must provide the full sequence of explicit reasoning segments $\mathcal{R}^{(i)}$, tool calls $u^{(i)}$, observations $o^{(i)}$, and the final answer a , from which TAC-CoT learns to replace each $\mathcal{R}^{(i)}$ with an implicit segment $C^{(i)}$ while preserving the surrounding explicit interactions.

We construct such data with two strategies depending on whether the source dataset already includes executable tool interactions, or only reasoning traces and gold answers.

- Sources with Tool Interactions. We convert the

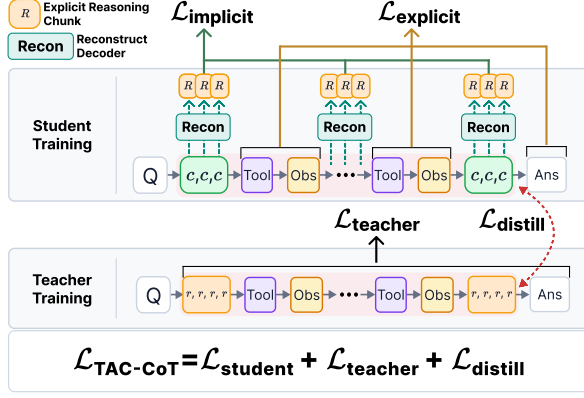


Figure 3: Training objective of TAC-CoT. The student generates continuous thought tokens, from which explicit reasoning chunks are reconstructed for supervision. Explicit outputs are trained with cross-entropy loss, while answer-boundary hidden states are distilled from an explicit-CoT teacher. The final objective combines student, teacher, and distillation losses.

provided executable annotations into tool calls. If the executable annotations are provided separately from the reasoning traces, we match their tool-call parameters and returned observations to the corresponding reasoning spans. We then normalize the matched interactions into a unified tool-call format with explicit tool names and parameters, and insert the tool calls and observations into the reasoning trajectory.

- *Sources without Tool Interactions.* As intermediate interactions are not directly available, we instead employ a stronger LLM (DeepSeek-V3; See Appendix A) as a tool-augmented solver to synthesize them. Given a question and the available tools, the model generates a complete trajectory with interleaved reasoning spans, tool calls, observations, and a predicted answer. We retain only trajectories whose predicted answers match the source gold answers, and normalize the retained trajectories into the same format.

2.3 Training Objective

Given the explicit trajectory τ_E from Section 2.2, training must teach the model to (i) encode the reasoning of each $\mathcal{R}^{(i)}$ into an implicit segment $C^{(i)}$, and (ii) preserve the surrounding tool calls $u^{(i)}$, observations $o^{(i)}$, and final answer a in explicit form. We achieve both through self-distillation (Shen et al., 2025), where the same backbone model serves as both the student and the teacher, with τ_E providing the teacher signal, as in Figure 3.

Student Training. Following the reconstruction idea of Wei et al. (2026), we attach an auxiliary

Task	Benchmark	Setting
Math	GSM8K (Cobbe et al., 2021)	In-domain
	GSM-Hard (Gao et al., 2023)	Out-of-domain
	SVAMP (Patel et al., 2021)	Out-of-domain
QA	HotpotQA (Yang et al., 2018)	In-domain
	2WikiMultihopQA (Ho et al., 2020)	Out-of-domain
	KILT-TriviaQA (Petroni et al., 2021)	Out-of-domain
Tool-use	ToolBench (Qin et al., 2024)	Tool-use evaluation

Table 1: Evaluation benchmarks for agentic mathematical reasoning, agentic QA and tool-use capability.

reconstruction decoder D_ϕ , which is discarded at inference. Because TAC-CoT hides only reasoning spans, the auxiliary decoder reconstructs the omitted reasoning content rather than the whole trajectory. Each $\mathcal{R}^{(i)}$ is split into b chunks ($R_1^{(i)}, \dots, R_b^{(i)}$), and each continuous thought token $c_t^{(i)}$ is aligned with $R_t^{(i)}$:

$$\mathcal{L}_{\text{implicit}} = - \sum_{i=1}^K \sum_{t=1}^B \log q_\phi(R_t^{(i)} | c_t^{(i)}), \quad (3)$$

where $q_\phi(R_t^{(i)} | c_t^{(i)})$ is the autoregressive reconstruction likelihood assigned by D_ϕ to $R_t^{(i)}$. This loss forces each implicit segment to encode the reasoning content that would otherwise be verbalized at the corresponding point in the tool-use trajectory.

For the model-generated explicit segments, including tool calls and final answers, we apply the standard cross-entropy loss $\mathcal{L}_{\text{explicit}}$. Observations are kept as explicit context returned by the environment. The student loss is

$$\mathcal{L}_{\text{student}} = \mathcal{L}_{\text{implicit}} + \mathcal{L}_{\text{explicit}}. \quad (4)$$

Teacher Training and Distillation. We train the teacher with the standard cross-entropy loss $\mathcal{L}_{\text{teacher}}$ on the explicit trajectory. We then distill the teacher’s hidden activations at the answer boundary so that the student after implicit reasoning matches the teacher after explicit reasoning:

$$\mathcal{L}_{\text{distill}} = \frac{1}{M} \sum_{l=1}^M \left\| \text{sg}[h_{\text{teacher}}^l] - h_{\text{student}}^l \right\|_1, \quad (5)$$

h_{teacher}^l and h_{student}^l are the teacher and student hidden activations at the colon token before answer generation, M is the number of layers, and $\text{sg}[\cdot]$ denotes the stop-gradient operation. The final objective is therefore $\mathcal{L}_{\text{TAC-CoT}} = \mathcal{L}_{\text{student}} + \mathcal{L}_{\text{teacher}} + \mathcal{L}_{\text{distill}}$.

Method	TU	Math								Question-Answering									
		In-domain		Out-of-domain				In-domain		Out-of-domain									
				GSM8K		GSM-Hard				SVAMP		Average		HotpotQA		2WikiMulti		TriviaQA	
		Acc↑	#Fwd↓	Acc↑	#Fwd↓	Acc↑	#Fwd↓	Acc↑	#Fwd↓	EM↑	#Fwd↓	EM↑	#Fwd↓	EM↑	#Fwd↓	EM↑	#Fwd↓	EM↑	#Fwd↓
Qwen2.5-0.5B-Instruct																			
Explicit-CoT	✗	50.9	117.3	13.4	160.0	59.0	61.9	41.1	113.1	13.2	951.2	21.0	739.1	11.3	891.6	15.2	860.6		
TokenSkip	✗	46.2	73.6	10.4	104.7	49.8	38.5	35.5	72.3	14.2	502.8	21.7	452.3	9.9	474.1	15.3	476.4		
PCCoT	✗	20.6	8.4	4.9	10.7	29.1	8.0	18.2	9.0	11.2	9.7	20.6	10.5	2.8	9.4	11.5	9.9		
CODI	✗	34.2	14.5	9.0	17.0	42.8	14.0	28.7	15.2	13.0	13.6	20.7	14.3	5.9	13.1	13.2	13.7		
SIM-CoT	✗	39.3	14.5	8.8	17.5	45.0	14.1	31.0	15.4	13.0	13.6	22.0	14.4	5.7	13.1	13.6	13.7		
TAC-CoT	✗	58.8	54.6	14.2	75.2	63.3	31.3	45.4	53.7	14.2	615.1	21.8	497.8	11.8	660.4	15.9	591.1		
Explicit-CoT	✓	52.4	109.2	30.4	145.7	62.4	58.4	48.4	104.4	33.1	309.0	40.8	247.8	53.3	293.0	42.4	283.3		
TAC-CoT	✓	61.2*	43.9	32.8	57.0	67.7†	26.4	53.9	42.4	36.2†	49.4	46.5*	39.8	54.8	43.3	45.8	44.2		
Llama-3.2-1B-Instruct																			
Explicit-CoT	✗	55.1	97.0	12.4	112.9	62.3	50.5	43.2	86.8	17.4	1126.1	19.8	928.5	30.9	957.6	22.7	1004.1		
TokenSkip	✗	52.8	57.2	11.6	67.9	60.3	28.1	41.6	51.1	17.1	438.8	22.0	361.3	24.0	387.1	21.0	395.7		
PCCoT	✗	45.6	7.5	10.8	7.9	53.9	7.5	36.8	7.6	4.6	11.3	7.9	11.9	4.8	10.6	5.8	11.3		
CODI	✗	55.4	15.0	12.9	17.2	61.3	15.4	43.2	15.9	18.8	13.4	22.9	14.1	24.8	13.0	22.2	13.5		
SIM-CoT	✗	56.0	15.0	12.6	20.5	61.2	16.5	43.3	17.3	19.1	13.4	23.8	14.1	26.5	12.9	23.1	13.5		
TAC-CoT	✗	61.9	45.3	14.3	52.6	69.0	26.9	48.4	41.6	18.3	592.7	22.8	554.6	28.8	512.4	23.3	553.2		
Explicit-CoT	✓	58.2	86.1	35.7	97.3	66.2	45.4	53.4	76.3	39.9	270.2	48.2	225.1	65.1	240.5	51.1	245.3		
TAC-CoT	✓	65.1*	38.2	39.3*	43.5	73.2*	23.6	59.2	35.1	41.4	52.8	52.0*	44.7	65.8	46.6	53.1	48.0		
Qwen3-4B																			
Explicit-CoT	✗	76.7	133.2	44.1	195.7	77.4	71.9	66.1	133.6	11.2	1496.0	10.6	1524.5	23.9	1273.7	15.2	1431.4		
TAC-CoT	✗	82.6	52.6	45.9	77.3	85.5	28.3	71.3	52.7	20.4	400.1	26.3	381.4	46.9	328.6	31.2	370.0		
Explicit-CoT	✓	78.0	115.2	49.1	154.3	78.5	61.7	68.5	110.4	43.4	272.2	59.9	224.5	72.3	235.3	58.5	244.0		
TAC-CoT	✓	83.4*	41.6	54.2*	55.9	86.0*	23.8	74.5	40.4	45.9*	43.1	60.5	44.9	75.5*	37.4	60.6	41.8		

Table 2: Main results on the math and QA benchmarks (Qwen2.5-0.5B, Llama-3.2-1B, and Qwen3-4B). #Fwd is the average number of forward passes per query. TU indicates whether tools are used. Best scores are highlighted in **bold**; * and † mark $p < 0.01$ and $p < 0.05$ improvements over the second-best via paired bootstrap with 10^5 samples (Koehn, 2004).

3 Experiments

We design experiments to evaluate two capabilities of our proposed method: (1) agentic math problem-solving training and evaluation to assess its reasoning capability, and (2) agentic QA training and evaluation to assess its agentic capability.

3.1 Experimental Setup

Training Datasets. TAC-CoT requires training to use implicit tokens. To evaluate mathematical reasoning, we use **GSM8K-Aug-NL** (Deng et al., 2024), which contains explicit interactions as textual explanations. Since GSM8K-Aug-NL contains explicit reasoning trajectories without executable interfaces, we insert executable calculator annotations from GSM8K-Aug (Deng et al., 2024). To evaluate QA capability, we use the **HotpotQA** (Yang et al., 2018) dataset. We construct 40,000 trajectories by using DeepSeek-V3 (DeepSeek-AI et al., 2025a) with the available tool and allowing it to solve each question through iterative tool interaction. We retain only trajectories whose final answer matches the gold annotation. Although teacher-generated trajectories may inherit teacher-specific biases, TAC-CoT and all

QA baselines are trained on the same trajectories; such biases are therefore shared across methods and cannot by themselves explain TAC-CoT’s relative improvements. Dataset construction details are provided in Appendix A.

Evaluation Benchmarks. We evaluate TAC-CoT on agentic math problem solving (Math), agentic Question-Answering (QA), and tool-use benchmarks, as summarized in Table 1. On tool-Bench, we use ‘Single’ for the I1 single-tool scenario, and ‘Multi’ as the I2 + I3 multi-tool scenario. We report accuracy (Acc) for math tasks and exact match (EM) for QA. To compare the inference cost independently of hardware and external API latency, we report the average number of forward passes (#Fwd) (Kuzina et al., 2026; Wei et al., 2026). For no-tool QA setting (TU = ✗), #Fwd also includes autoregressive generation of missing tool observations; see Appendix B.2.

Implementation Details. We use Qwen2.5-0.5B-Instruct (Yang et al., 2024), Llama-3.2-1B-Instruct (Grattafiori et al., 2024), and Qwen3-4B (Yang et al., 2025) as backbone models. For agentic QA, we use the Wikipedia API as the search

engine, following Yao et al. (2022). Further details are provided in Appendix B. We provide pseudocode for reproducibility in Figure 10. For TAC-CoT, we set $B = 3$. Ablation results for different B values are provided in Appendix C.1.

Baselines. We compare TAC-CoT against (1) an **Explicit-CoT** agent, which serves as a strong baseline for tool use and reasoning by using full explicit reasoning traces, fine-tuned on the same training trajectories as our method; (2) **TokenSkip** (Xia et al., 2025), an explicit CoT method that compresses intermediate reasoning trajectories by pruning low-importance tokens; (3) **PC-CoT** (Wu et al., 2025), which updates implicit thought tokens in parallel using Jacobi iteration, improving efficiency over sequential implicit CoT; (4) **CODI** (Shen et al., 2025), an implicit-CoT method trained via trajectory-level self-distillation from CoT-based teacher signals; and (5) **SIM-CoT** (Wei et al., 2026), which extends CODI by adding step-level supervision to reconstruct reasoning steps from latent embeddings.

3.2 Main Results

Our evaluation examines whether TAC-CoT achieves its design goals in addressing the limitations of extending existing CoT approaches to agentic systems, focusing on four aspects: tool-use effectiveness, internal reasoning ability, query-adaptive continuous-thought budgeting, and inference efficiency. Table 2 presents the main results on agentic math and QA benchmarks across tool-calling and no-tool settings.

Performance on Agentic Reasoning. Overall, TAC-CoT consistently outperforms Explicit-CoT and the other baselines. Averaged over all backbone models, Our method improves math accuracy by 5.8%p and QA performance by 2.4 points compared with Explicit-CoT. The gains also transfer to out-of-domain benchmarks: Ours improves math performance by 6.5%p in-domain and 5.5%p out-of-domain, and QA performance by 2.4 points in-domain and 2.5 points out-of-domain. This consistent trend suggests that TAC-CoT provides robust adaptive implicit reasoning beyond the training-domain distribution.

To isolate the effect of implicit reasoning from external tool execution, we compare TAC-CoT with other baselines in the no-tool setting on math datasets. While TokenSkip, which shortens rationales by pruning tokens, underperforms

Method	HotpotQA		2WikiMulti		TriviaQA	
	@5	@10	@5	@10	@5	@10
Explicit-CoT	48.8	51.7	70.9	71.6	72.6	75.5
TAC-CoT	51.8[†]	55.4*	73.7*	74.4[†]	74.5	77.4

Table 3: Answer Coverage @ k (%) over retrieval steps on the QA benchmarks, Qwen2.5-0.5B-Instruct. All AC values are reported as percentages; Best scores are **bold**; * and [†] mark $p < 0.01$ and $p < 0.05$ improvements over the underlined Explicit-CoT.

Explicit-CoT, TAC-CoT consistently outperforms the Explicit-CoT agent across all math datasets.

These results indicate that the improvement in internal reasoning does not simply come from shorter reasoning traces. Consistently, TAC-CoT also outperforms baselines that either remove natural-language reasoning entirely or retain concise, explicit reasoning with tool use; see Appendix C.2.

Effectiveness of TAC-CoT. The effectiveness of TAC-CoT comes from both tool-use and query-adaptive implicit reasoning. Tool-use substantially improves QA performance, yielding up to a 29.9 point average gain over the no-tool variant while reducing forward passes by $13.3\times$ on Qwen2.5-0.5B. Even without tools, TAC-CoT benefits from adaptive budgeting: on math benchmarks, its no-tool variant outperforms prior implicit-CoT baselines while requiring far fewer forward passes than explicit CoT. These results show that TAC-CoT improves efficiency and performance by adaptively allocating continuous reasoning, with additional gains when external knowledge is needed.

Supplementary full fine-tuning results are provided in Appendix C.3.

3.3 Diagnostic Analysis

We conduct diagnostic analyses to identify the sources of TAC-CoT’s performance gains across four dimensions: tool-use quality, reduced error accumulation, query-adaptive budgeting efficiency, and lower inference-token cost.

Effectiveness of Tool Use. We evaluate whether TAC-CoT improves tool-mediated decisions on QA benchmarks using Answer Coverage@ k (AC@ k), which measures whether the gold answer appears in the first k retrieved observations. As shown in Table 3, TAC-CoT improves AC@5 and AC@10 across all QA datasets, with significant gains on HotpotQA and 2WikiMultiHopQA. Notably, its AC@5 is comparable to the AC@10

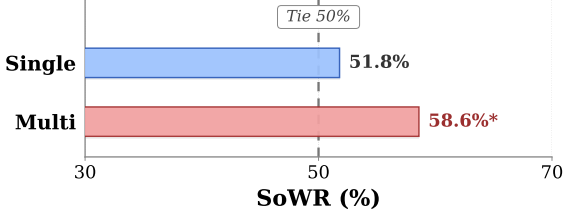


Figure 4: Solvable win rate (SoWR) of TAC-CoT against Explicit-CoT on ToolBench, judged by GPT-5.1. ‘Single’ denotes the single-tool scenario, and ‘Multi’ denotes the multi-tool scenario. * marks $p < 0.01$.

Metric	E-CoT	# Replaced				Ours
		1	2	3	4	
Mean Acc $\uparrow$	41.1	41.7	42.4	43.4	44.9	45.4
# Fwd $\downarrow$	113.1	96.2	79.2	63.4	58.9	53.7

Table 4: Effect of replacement depth on Qwen2.5-0.5B-Instruct average accuracy over GSM8K, GSM-Hard, and SVAMP. All accuracy values are reported as percentages. **E-CoT** denotes the Explicit-CoT agent, and **# Replaced** denotes the number of explicit reasoning segments swapped for continuous thought tokens.

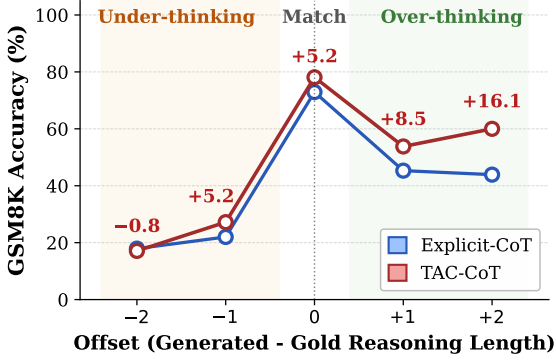


Figure 5: GSM8K accuracy stratified by the offset between the number of generated reasoning steps and the gold step count in the gold solution (negative = under-thinking, 0 = matched depth, positive = over-thinking).

of Explicit-CoT, indicating that TAC-CoT retrieves supporting evidence in fewer steps. We further analyze tool-use complexity on ToolBench by fine-tuning Qwen2.5-0.5B-Instruct and comparing TAC-CoT with Explicit-CoT under the StableToolBench evaluation protocol (Guo et al., 2024). As shown in Figure 4, TAC-CoT achieves Solvable Win Rates of 51.8% on the single-tool and 58.6% in the multi-tool scenario, with the latter gain being statistically significant. This suggests that interleaved implicit reasoning is especially beneficial for complex multi-tool interactions. A detailed analysis is provided in Appendix C.4.

Error Accumulation Mitigation. Table 2 shows that TAC-CoT improves internal reasoning even

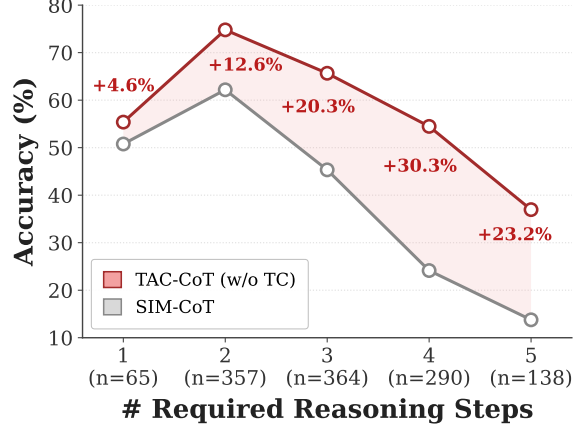


Figure 6: Accuracy comparison across GSM8K subsets grouped by the number of required reasoning steps. Each n denotes the number of samples in the corresponding reasoning-step group.

without external execution. To test whether this comes from reduced error accumulation, we analyze reasoning compression and length mismatch. First, we progressively replace explicit reasoning segments with implicit ones when training Qwen2.5-0.5B-Instruct on the augmented GSM8K trajectories. We vary the replacement depth from one to four segments, since four corresponds to the 90th percentile of the number of explicit reasoning segments in the training trajectories. As shown in Table 4, average accuracy on GSM8K, GSM-Hard, and SVAMP increases monotonically as more segments become implicit, suggesting that implicit CoT mitigates errors from lengthy rationales. Second, following the step-efficiency analysis of prior work (Wu et al., 2026), we group GSM8K examples by the offset between generated and gold reasoning lengths. Figure 5 shows that both Explicit-CoT and TAC-CoT perform best under matched lengths and degrade under over-thinking offsets. However, TAC-CoT degrades less, indicating that implicit reasoning reduces the penalty from unnecessary intermediate steps. We discuss further analysis of error accumulation in Appendix C.5.

Query-Adaptive Continuous-Thought Budget.

Unlike prior implicit-CoT methods that use a fixed continuous-token budget for every query, TAC-CoT adaptively allocates its total reasoning budget according to the query. This query-adaptive continuous-thought budgeting contributes to the performance gains, especially for math tasks that depend on internal reasoning. As shown in Figure 6, the benefit becomes larger as the required number of reasoning steps increases.

TAC-CoT’s query-adaptive budgeting benefits inputs that require different amounts of internal deliberation before producing the final answer. We group GSM8K examples into five subsets according to the number of gold reasoning steps, ranging from one to five steps, and treat this number as the required reasoning depth. We then evaluate each subset separately and compare TAC-CoT without tool use against SIM-CoT, a state-of-the-art fixed-budget implicit-CoT baseline, using Qwen2.5-0.5B-Instruct. Figure 6 shows that TAC-CoT consistently outperforms SIM-CoT across all reasoning-depth groups, with larger gains on examples that require more reasoning steps. The improvement increases from 4.6 points on one-step examples to 30.3 points on four-step examples. This trend suggests that assigning the same continuous-thought-token budget to every input becomes increasingly restrictive as the required number of reasoning steps grows. In contrast, TAC-CoT increases the total continuous-token budget by continuing through additional implicit reasoning segments when more reasoning steps are needed. These results demonstrate that TAC-CoT flexibly allocates reasoning budget across queries, which is especially important for reasoning-intensive tasks.

Inference Efficiency. TAC-CoT substantially reduces inference cost relative to Explicit-CoT while improving task accuracy. Across backbones, it reduces the per-query forward count by up to $2.6\times$ on math benchmarks and $6.8\times$ on QA benchmarks, with larger savings on QA due to longer multi-step trajectories. This suggests that TAC-CoT becomes increasingly beneficial for longer trajectories.

Although fixed-budget implicit-CoT baselines use fewer forward passes, they do not adapt to query complexity or retrieve missing observations. This limitation is especially harmful for QA, where answers often depend on retrieved evidence, leading to pronounced performance degradation. In contrast, TAC-CoT achieves a better cost–performance trade-off by reducing inference cost relative to Explicit-CoT while avoiding the QA degradation of fixed-budget implicit-CoT methods.

On a single NVIDIA L40S GPU, TAC-CoT achieves measured wall-clock speedups of $2.3\times$ on math and $2.4\times$ on QA. Since realized latency depends on hardware, implementation, and external API overhead, #Fwd reductions should not be interpreted directly as wall-clock speedups; see Appendix C.6 for full results.

4 Related Works

Explicit and Implicit CoT. Chain-of-Thought (CoT) improves the reasoning ability of large language models by generating intermediate reasoning before the final answer (Wei et al., 2022; Kojima et al., 2022). CoT methods are commonly categorized as explicit or implicit, depending on whether intermediate reasoning is represented as discrete tokens or continuous representations. Explicit CoT is readable and easy to supervise (Wei et al., 2022; Kojima et al., 2022; Yao et al., 2023), but decoding long reasoning traces increases inference cost (Han et al., 2025; Xia et al., 2025) and exacerbates stepwise error accumulation (Wu et al., 2026).

Implicit CoT instead performs intermediate reasoning in continuous latent space (Hao et al., 2025; Shen et al., 2025; Wu et al., 2025; Wei et al., 2026; Kuzina et al., 2026). For example, Coconut feeds hidden states back as continuous thought tokens (Hao et al., 2025), CODI distills explicit CoT into continuous representations (Shen et al., 2025), PCCoT updates latent thought tokens in parallel for efficient inference (Wu et al., 2025), and SIM-CoT stabilizes implicit reasoning via step-level reconstruction (Wei et al., 2026). Although these methods reduce the cost of verbalized reasoning, they are mainly designed for standalone reasoning: they typically allocate a fixed number of continuous thought tokens per query and do not expose intermediate decisions as executable interfaces, limiting their suitability for agentic settings that require repeated reasoning and acting.

In contrast, TAC-CoT extends implicit CoT to tool-using agentic systems by interleaving compact latent reasoning segments with explicit tool interactions, enabling tool calling and query-adaptive reasoning budgets. At the same time, compared with explicit-CoT agents, TAC-CoT avoids the need to verbalize intermediate reasoning, thereby reducing inference cost and mitigating error accumulation from lengthy reasoning traces.

Tool-Using LLM Agents. LLMs are increasingly studied as agents that solve tasks through interleaved reasoning, tool calls, and observations (Yao et al., 2022; Schick et al., 2023; Shinn et al., 2023; Yang et al., 2025). ReAct formalizes this reasoning–acting loop (Yao et al., 2022), while Toolformer (Schick et al., 2023) and ART (Paranjape et al., 2023) show that models can invoke external APIs and resume generation after tool exe-

cution. However, most tool-using agents still rely on explicit CoT (Wei et al., 2022; Kojima et al., 2022; Yao et al., 2023), whose repeated generation across multi-step interaction loops makes inference costly (Han et al., 2025; Xia et al., 2025; Aytes et al., 2025) and vulnerable to error accumulation (Wu et al., 2026).

5 Conclusion

We presented TAC-CoT, a tool-empowered adaptive continuous CoT framework that extends implicit reasoning to tool-using agentic problem solving. By interleaving compact continuous reasoning segments with explicit tokens for tool calls, observations, and final answers, TAC-CoT preserves the executable interface required for agentic workflows while avoiding the cost of fully verbalized reasoning traces. Across agentic math and knowledge-intensive QA benchmarks, TAC-CoT consistently outperforms Explicit-CoT agents in both in-domain and out-of-domain settings, while using substantially fewer model forward passes. Further analyses show that TAC-CoT improves tool-use decisions, performs better in multi-tool scenarios, mitigates error accumulation, and benefits from query-adaptive continuous thought-token allocation.

Limitations

This work has a few limitations: First, the adaptive routing mechanism operates at the segment level rather than the token level. Second, longer interaction horizons remain an important direction for evaluating the practical deployment of TAC-CoT.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00334343). Additionally, this work was supported by the National IT Industry Promotion Agency (NIPA) grant funded by the Korea government (MSIT) (No. RS-2026-25621604).

References

Pranjal Aggarwal, Seungone Kim, Jack Lanchantin, Sean Welleck, Jason E Weston, Ilia Kulikov, and Swarnadeep Saha. 2026. [Optimalthinkingbench: Evaluating over and underthinking in LLMs](#). In *The Fourteenth International Conference on Learning Representations*.

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. [Sketch-of-thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 24296–24320, Suzhou, China. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *Preprint*, arXiv:2110.14168.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025a. [Deepseek-v3 technical report](#). *Preprint*, arXiv:2412.19437.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025b. [Deepseek-v3.2: Pushing the frontier of open large language models](#). *Preprint*, arXiv:2512.02556.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. 2024. [Implicit chain of thought reasoning via knowledge distillation](#).

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [Pal: program-aided language models](#). In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. [The llama 3 herd of models](#). *arXiv preprint arXiv:2407.21783*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. [StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156, Bangkok, Thailand. Association for Computational Linguistics.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. [Token-budget-aware LLM reasoning](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855, Vienna, Austria. Association for Computational Linguistics.

- Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. 2025. [Training large language models to reason in a continuous latent space](#). In *Second Conference on Language Modeling*.
- Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. [Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps](#). In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.
- Kyochul Jang, Donghyeon Lee, Kyusik Kim, Dongseok Heo, Taewhoo Lee, Woojeong Kim, and Bongwon Suh. 2025. [DICE-BENCH: Evaluating the tool-use capabilities of large language models in multi-round, multi-party dialogues](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26822–26846, Vienna, Austria. Association for Computational Linguistics.
- Lars Benedikt Kaesberg, Jonas Becker, Jan Philip Wahle, Terry Ruas, and Bela Gipp. 2025. [Voting or consensus? decision-making in multi-agent debate](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11640–11671, Vienna, Austria. Association for Computational Linguistics.
- Jiin Kim, Byeongjun Shin, Jinha Chung, and Minsoo Rhu. 2026. [The cost of dynamic reasoning: Demystifying ai agents and test-time scaling from an ai infrastructure perspective](#). *Preprint*, arXiv:2506.04301.
- Philipp Koehn. 2004. [Statistical significance tests for machine translation evaluation](#). In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. [Large language models are zero-shot reasoners](#). In *Advances in Neural Information Processing Systems*.
- Anna Kuzina, Maciej Pióro, and Babak Ehteshami Bejnordi. 2026. [Kava: Latent reasoning via compressed KV-cache distillation](#). In *The Fourteenth International Conference on Learning Representations*.
- Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. [Art: Automatic multi-step reasoning and tool-use for large language models](#). *Preprint*, arXiv:2303.09014.
- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. [Are NLP models really able to solve simple math word problems?](#) In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.
- Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. [KILT: a benchmark for knowledge intensive language tasks](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, Online. Association for Computational Linguistics.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. [ToolLLM: Facilitating large language models to master 16000+ real-world APIs](#). In *The Twelfth International Conference on Learning Representations*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. 2025. [CODI: Compressing chain-of-thought into continuous space via self-distillation](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 677–693, Suzhou, China. Association for Computational Linguistics.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. [Reflexion: language agents with verbal reinforcement learning](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Alexander von Recum, Leander Gierbach, and Zeynep Akata. 2026. [Are reasoning LLMs robust to interventions on their chain-of-thought?](#) In *The Fourteenth International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. [Chain of thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*.
- Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. 2026. [SIM-cot: Supervised implicit chain-of-thought](#). In *The Fourteenth International Conference on Learning Representations*.
- Haoyi Wu, Zhihao Teng, and Kewei Tu. 2025. [Parallel continuous chain-of-thought with jacobi iteration](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 914–926, Suzhou, China. Association for Computational Linguistics.

- Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2026. [When more is less: Understanding chain-of-thought length in LLMs](#). In *The Fourteenth International Conference on Learning Representations*.
- Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. [TokenSkip: Controllable chain-of-thought compression in LLMs](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3351–3363, Suzhou, China. Association for Computational Linguistics.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. [Qwen2.5 technical report](#).
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. [HotpotQA: A dataset for diverse, explainable multi-hop question answering](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023. [Tree of thoughts: Deliberate problem solving with large language models](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. [React: Synergizing reasoning and acting in language models](#). *arXiv preprint arXiv:2210.03629*.
- Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. 2023. [Exchange-of-thought: Enhancing large language model capabilities through cross-model communication](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153, Singapore. Association for Computational Linguistics.
- Qizheng Zhang, Michael Wornow, and Kunle Olukotun. 2026. [Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

A Details of Data Construction

This section provides the details of the recipe described in Section 2.2 for each domain.

Math Problem Solving. The math domain follows the source-obtainable interaction setting described. For GSM8K, where data with explicit interactions already exist, we use GSM8K-Aug and GSM8K-Aug-NL, two GSM8K augmentations with 385k samples each (Deng et al., 2024). GSM8K-Aug retains only the arithmetic equations in each reasoning step (e.g., «86.7*15*.01=13.005»), whereas GSM8K-Aug-NL contains only explicit reasoning (e.g. ... day, 86.7 x 15% = 13.005 coins...). Because the two datasets have the same questions, we align examples by question and insert the equations from GSM8K-Aug into the corresponding natural language reasoning trajectory from GSM8K-Aug-NL.

Question-Answering. For HotpotQA, which provides only questions and their gold answers without faithful tool-use trajectories, we construct trajectories by prompting DeepSeek-V3 to solve each question with access to a retrieval tool. Following Yao et al. (2022), we implement a Wikipedia retrieval system and use the ReAct-style few-shot demonstrations from their setup. The prompt template is shown in Figures 11 and 12.

Quality Audit and Construction Cost. To assess the quality of the synthesized QA trajectories, we randomly sampled 200 trajectories and evaluated them using GPT-5.1, a model different from the DeepSeek-V3 teacher. The evaluator assessed whether the generated actions and final answers were logically valid, and judged 99% of the sampled trajectories to be valid.

Generating the QA trajectories cost \$102.73 in total, including generations discarded during answer-based filtering. The filtering pass rate was 56.24%. Because exact-match filtering may reject semantically correct answers expressed differently from the gold annotation, this procedure may underestimate the usable trajectory yield.

B Implementation Details

B.1 Training hyperparameters

For the CODI loss weight, we set $\alpha_{\text{CODI}} = 1$ for TAC-CoT and $\alpha_{\text{CODI}} = 10$ for PCCoT, CODI, and SIM-CoT, following Shen et al. (2025); Wu et al. (2025); Wei et al. (2026). For TokenSkip,

we use a 50% compression ratio during training to keep its number of forward passes comparable to that of TAC-CoT. For the larger Qwen3-4B backbone, we use $B = 2$ continuous tokens per segment for a practical training setup.

B.2 Implicit CoT Baseline Details

For each implicit CoT baseline, we report the higher score between models trained on GSM8K-Aug-NL and GSM8K-Aug.

Evaluation benchmark. For math benchmarks, we evaluate on the full test sets. For QA benchmarks, we evaluate on 1,000 randomly sampled examples from each dataset. This follows approaches used by Yin et al. (2023); Kaesberg et al. (2025); Jang et al. (2025). Specifically, we follow the standard sample-size estimate for population proportions:

$$n_0 = \frac{Z^2 \cdot p \cdot (1 - p)}{d^2},$$

where $Z = 1.96$, $p = 0.5$, and $d = 0.05$, corresponding to a 95% confidence level and a 5% margin of error under the conservative maximum-variance assumption. This gives $n_0 \approx 384$, so our choice of 1,000 examples per QA dataset is above the required sample size. Since EM and Answer Coverage are averaged binary outcomes over examples, 1,000 random samples yield a conservative worst-case 95% margin of error of approximately $1.96\sqrt{0.25/1000} = 3.1$ percentage points before finite-population correction.

Calculator setting. For math tasks, we build a calculator database by extracting equations from the training trajectories. During evaluation, if an expression matches an entry in this database, we use the corresponding stored result. Otherwise, we process the expression before calculation and then execute it with the calculator. This design handles cases where expressions contain units or other non-numeric tokens that cannot be directly evaluated by a standard calculator.

ToolBench API details. For ToolBench, many of the original APIs are no longer available at inference time. Following Guo et al. (2024), we therefore use MirrorAPI. Because TAC-CoT does not verbalize its reasoning, we explicitly note this in the GPT-5.1 judge prompt.

No-tool evaluation protocol. For the no-tool setting (TU = ✗), we disable the external tool executor

Hyperparameter	Qwen2.5 0.5B-Instruct	Llama-3.2 1B-Instruct	Qwen3 4B
Fine-tuning strategy	Full fine-tuning	LoRA	LoRA
LoRA rank	–	128	128
LoRA scaling factor	–	32	32
LoRA dropout	–	0.1	0.1
Learning rate	1×10^{-4}	8×10^{-4}	1×10^{-4}
LR scheduler	Cosine	Cosine	Cosine
Warmup	0.03	0.03	0.03
Optimizer	AdamW	AdamW	AdamW
Batch size	128	128	128
Gradient norm clipping	2	2	2
Training epochs	10	10	5

Table 5: Training hyperparameters used for each backbone model. Values are shared across all compared methods unless otherwise specified.

#CT	Math			
	GSM8K	GSM-Hard	SVAMP	Average
0	55.7	13.9	56.3	42.0
1	58.5	14.7	61.5	44.9
3	58.8	14.2	63.3	45.4

Table 6: Effect of the number of continuous thought tokens (#CT) per continuous segment on math datasets, using Qwen2.5-0.5B-Instruct. All scores are reported as percentages.

while preserving the trajectory format used by tool-using agents. When the model emits an action-like span, no environment-returned observation is injected. Instead, the model continues generation and produces the corresponding observation text by itself. Thus, the no-tool setting evaluates reasoning without externally supplied tool outputs. All generated observation tokens are counted in the inference cost, and we use the same answer extraction and evaluation metrics as in the tool-use setting.

C Additional Experiments

This appendix provides additional analyses that complement the main experimental results. We first study how the number of allowed tool calls can mitigate error propagation (Yao et al., 2022). We then analyze how errors accumulate as reasoning depth increases, which motivates the need for shorter reasoning methods, such as implicit CoT. Finally, we report supplementary full fine-tuning results for additional backbone settings that are not included in the main results table.

C.1 Effect of Continuous-Thought Tokens

We analyze the effect of the number of continuous thought tokens (CT) per continuous segment. We vary the CT budget over $\{0, 1, 3\}$ and evaluate Qwen2.5-0.5B-Instruct on GSM8K, GSM-Hard, and SVAMP. Table 6 shows the results. Compared to the 0 CT setting, using CT tokens improves the average accuracy from 42.0% to 44.9% with one CT token and 45.4% with three CT tokens. These results suggest that allocating more CT can be beneficial.

C.2 Controlled Baseline Comparisons

We conduct additional controlled comparisons to distinguish the benefit of implicit reasoning from two simpler explanations: removing natural-language rationales and shortening explicit reasoning traces. All experiments in this section use Qwen2.5-0.5B-Instruct.

No Reasoning. We first consider a no reasoning control. This baseline removes natural-language rationales, while retaining the arithmetic equations used for intermediate computation. This comparison isolates whether TAC-CoT’s improvement can be explained simply by removing verbose natural-language reasoning. As shown in Table 7, TAC-CoT improves the average accuracy from 42.0% to 45.4% in the no-tool setting.

Concise Explicit-CoT with Tools. We further compare against a concise explicit-CoT agent with the same tool-enabled setting. Specifically, we extend TokenSkip to tool use and apply the same 50% compression ratio used in our main experiments, thereby shortening explicit reasoning traces while preserving explicit reasoning and tool interaction. TAC-CoT achieves an average accuracy of 53.9%, compared with 45.4% for TokenSkip with tools. The improvement is consistent across GSM8K, GSM-Hard, and SVAMP.

Overall, these controlled comparisons indicate that TAC-CoT’s improvement is not solely attributable to eliminating natural-language prose or reducing the length of explicit reasoning.

C.3 Additional Full Fine-Tuning Results

We report supplementary results under full fine-tuning for backbone settings not covered in the main results table. Table 8 shows results for Llama-3.2-1B-Instruct with full fine-tuning, complementing the LoRA-adapted Llama results in Table 2.

Method	GSM8K	GSM-Hard	SVAMP	Average
<i>No-tool setting</i>				
No Reasoning	55.7	13.9	56.3	42.0
TAC-CoT	58.8	14.2	63.3	45.4
<i>Tool-enabled setting</i>				
TokenSkip	52.3	27.0	57.0	45.4
TAC-CoT	61.2	32.8	67.7	53.9

Table 7: Controlled comparisons on the mathematical reasoning benchmarks using Qwen2.5-0.5B-Instruct. The no-natural-language reasoning control removes natural-language rationales and continuous thought tokens while retaining arithmetic equations. TokenSkip + Tool provides a concise explicit-CoT control under the tool-enabled setting. All scores are accuracy (%).

Method	Math			
	GSM8K	GSM-Hard	SVAMP	Average
Explicit-CoT	52.2	13.9	68.5	44.9
TAC-CoT	62.0	14.9	68.7	48.5

Table 8: Math accuracy of Explicit-CoT agent and TAC-CoT on GSM8K, GSM-Hard, and SVAMP when Llama-3.2-1B-Instruct is fully fine-tuned (no LoRA), reported for reference against the LoRA-adapted Llama numbers in Table 2.

Table 9 reports full fine-tuning results for a GPT-2 backbone. Across both settings, TAC-CoT consistently improves over the Explicit-CoT agent baseline, further supporting the effectiveness of our method across model families and settings.

C.4 Impact of the Tool-Use Budget

To investigate how the number of permitted tool-use affects performance, we conduct an ablation study on GSM8K samples containing four or more equations, since each equation corresponds to a potential calculator invocation. We vary the maximum number of calculator calls from 0 to 4, and also evaluate with unlimited calls. As shown in Figure 8, accuracy improves monotonically as the number of allowed tool calls increases. This suggests that each additional tool call can contribute to mitigate error propagation during multi-step reasoning.

C.5 Impact of Error Accumulation

As shown in Figure 7, we trained GPT-2 and Llama-3.2-1B-Instruct on GSM8k-Aug and GSM8K-Aug-NL to analyze how intermediate reasoning errors accumulate as reasoning depth increases. In these datasets, equations are inserted into explicit reasoning. When gold reasoning is provided, GSM8K-

Method	Math			
	GSM8K	GSM-Hard	SVAMP	Average
Explicit-CoT	32.8	6.6	38.2	25.9
TAC-CoT	47.7	10.8	45.0	34.5

Table 9: Math accuracy of Explicit-CoT agent and TAC-CoT on GSM8K, GSM-Hard, and SVAMP under a GPT2 backbone with full fine-tuning, complementing the Qwen and Llama backbones reported in Table 2.

Method	Dataset	# Fwd	ms / sample
<i>Agentic Math</i>			
TAC-CoT	GSM8K	43.9	545.0
	GSM-Hard	57.0	712.2
	SVAMP	26.2	335.9
	Average	42.4	531.0
Explicit-CoT	GSM8K	109.2	1232.2
	GSM-Hard	145.7	1673.5
	SVAMP	58.4	682.5
	Average	104.4	1196.1
<i>Agentic QA</i>			
TAC-CoT	HotpotQA	49.4	4455.3
	2WikiMulti	39.8	2409.6
	TriviaQA	43.3	2697.7
	Average	44.2	3187.5
Explicit-CoT	HotpotQA	272.2	7765.8
	2WikiMulti	224.5	7372.7
	TriviaQA	235.3	7825.0
	Average	244.0	7654.5

Table 10: Inference-time comparison between TAC-CoT and Explicit-CoT on the Qwen2.5-0.5B-Instruct backbone. **ms/sample** denotes the average wall-clock inference latency in milliseconds per sample, measured on a single NVIDIA L40S GPU.

Aug-NL-trained models outperform models trained on GSM8K-Aug, which are not trained to generate natural-language rationales. However, this advantage progressively diminishes as the number of generated reasoning steps increases. This trend suggests that, although explicit natural language reasoning can be beneficial under oracle reasoning conditions, error accumulation during intermediate steps can degrade accuracy. The degradation is less pronounced for the larger 1B model, which is consistent with prior observations that larger models are more robust to error accumulation in multi-step reasoning (von Recum et al., 2026).

C.6 Wall-Clock Speedup of TAC-CoT

We additionally report wall-clock inference time on a single NVIDIA L40S GPU using Qwen2.5-

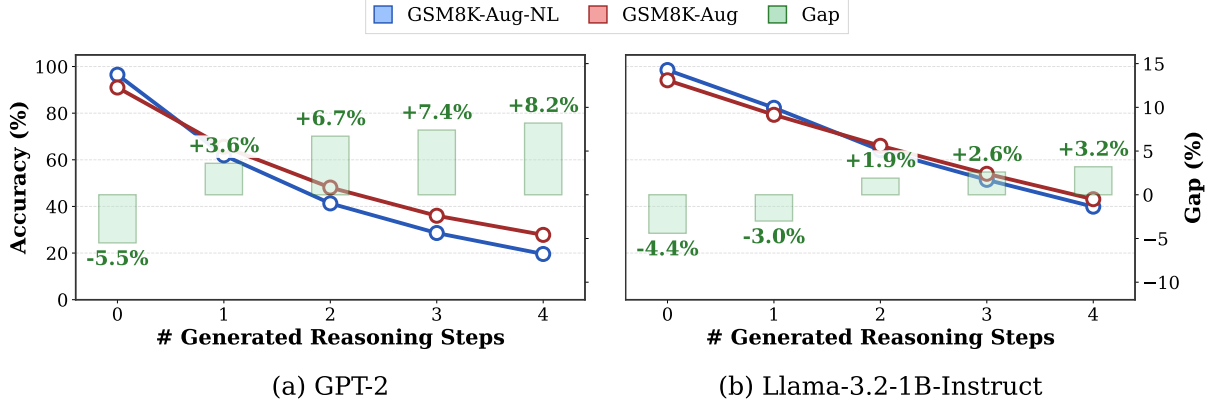


Figure 7: Effect of reasoning depth on error accumulation.

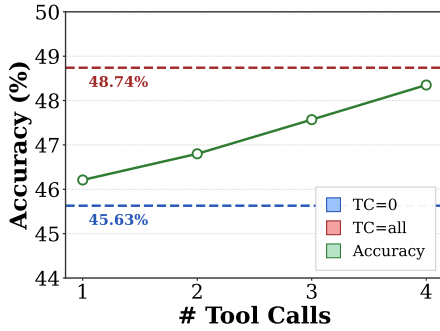


Figure 8: Impact of the number of tool calls on GSM8K accuracy. TC=0 denotes no tool calling, and TC=all denotes unlimited tool calls.

0.5B-Instruct. As shown in Table 10, TAC-CoT consistently reduces both the number of forward passes and end-to-end latency compared with the Explicit-CoT agent. On average, TAC-CoT uses $2.3\times$ fewer forward passes on the math benchmarks and $5.5\times$ fewer forward passes on the QA benchmarks. These reductions also translate into practical wall-clock speedups: TAC-CoT runs $2.3\times$ faster on math benchmarks and $2.4\times$ faster on QA benchmarks on average. When isolating model generation time by excluding external overheads, the speedup further increases, reaching up to $3.8\times$.

These speedups are smaller than the reductions in forward passes because forward-pass count does not translate linearly into end-to-end latency. In our current implementation, total latency still includes non-optimized execution paths, parallelization overheads, and API-call latency. Thus, while TAC-CoT already provides consistent practical speedups in wall-clock time, the gap between forward-pass reduction and realized latency reduction suggests that further gains are possible with more optimized implementation and execution-

level parallelization.

D Case Studies

Figure 9 presents qualitative examples from agentic math and QA using Qwen2.5-0.5B-Instruct. Explicit-CoT and TAC-CoT are both tool-using agents, while SIM-CoT does not expose intermediate tool interactions.

In the math example, Explicit-CoT calls the calculator but applies the overtime rate to all 45 hours instead of the 5 overtime hours. SIM-CoT also fails without explicit intermediate computation. TAC-CoT grounds each arithmetic sub-step with calculator calls and preserves the correct decomposition of regular pay, overtime hours, and final summation.

In the QA example, SIM-CoT hallucinates “Wake Island” without retrieval. Explicit-CoT searches for evidence but follows an irrelevant O’ahu suggestion and later makes a false factual claim. TAC-CoT instead uses the failed search’s similar-results list to identify “Laysan” and then retrieves confirming evidence.

E Code Details

Figure 10 presents the TAC-CoT inference loop code. We describe the key implementation decisions below.

Continuous thought token injection. The continuous thought token $c_t \in \mathbb{R}^d$ is produced by applying a projection head Proj_θ to the last hidden state of the transformer: $c_t = \text{Proj}_\theta(h_t)$, where h_t is extracted from the final layer at the last sequence position. The continuous token is fed back to the model via `inputs_embeds`, bypassing the vocabulary embedding lookup entirely. No vocabulary index is ever assigned to a continuous thought

token.

Route selection and observation injection. At the end of each latent segment, c_B is fed via `inputs_embeds` to obtain route logits. If the `[[ANSWER]]` logit exceeds $\ell_{[[EXPLICIT]]}$, the model switches to answer generation; otherwise, action tokens are generated greedily until a stop token (`</action>`). Upon a tool call, the closing action token and the returned observation text are concatenated and processed in a single forward pass, after which the latent iteration loop restarts.

(a) Agentic Math (GSM8K)

Q: Eliza’s rate per hour for the first 40 hours she works each week is \$10. She also receives an overtime pay of 1.2 times her regular hourly rate. If Eliza worked for 45 hours this week, how much are her earnings for this week? **Gold: \$460**

Gold Trajectory

Eliza is entitled to $45 - 40 = 5$ hours overtime pay. Her hourly rate for the overtime pay is $\$10 \times 1.2 = \$12$. So, Eliza will receive $\$12 \times 5 = \$60$ for overtime pay. Her regular weekly earning is $\$10 \times 40 = \$400$. Thus, Eliza will receive a total of $\$400 + \$60 = \$460$ for this week’s work.

Explicit-CoT (SFT)

[×] \$940

Eliza earns $\$10 \times 40 = \$400$ for the regular hours she worked. The overtime pay she receives is $\$10 \times 1.2 = \$12$ per hour. So, she earns $\$12 \times 45 = \$540$ for the overtime. ←should be 5 h
Therefore, Eliza earns $\$400 + \$540 = \$940$ for this week. The answer is: **940**

SIM-CoT

[×] 210

[L×6] [[ANSWER]] The answer is: **210**

TAC-CoT (Ours)

[✓] \$460

$\llbracket 10 \times 40 = 400 \rrbracket$ [L×3] $\llbracket 1.2 \times 10 = 12 \rrbracket$ [L×3] $\llbracket 12 \times 5 = 60 \rrbracket$ [L×3] $\llbracket 400 + 60 = 460 \rrbracket$ [L×3] [[ANSWER]] The answer is: **460**

(b) Agentic QA (HotpotQA)

Q: What island is located 808 nmi northwest of the capital of Hawaii?

Gold: Laysan

Gold Answer Laysan (No gold trajectory available for HotpotQA.)

Explicit-CoT (SFT)

[×] O’ahu

Thought 1: I need to find the capital of Hawaii first.
Search[Hawaii capital] → Obs. 1: *Could not find. Similar: [... ‘Honolulu (redirect from Capital of Hawaii)’...]*
Thought 2: The capital of Hawaii is Honolulu. Find an island 808 nmi northwest of Honolulu.
Search[Islands near Honolulu] → Obs. 2: *Could not find. Similar: [... ‘O’ahu (redirect from Island of O’ahu)’...]*
Thought 3: O’ahu is mentioned. Let me check O’ahu.
Search[O’ahu] → Obs. 3: *... O’ahu is 44 miles long from east to west. ...*
Thought 4–8: Multiple failed searches for the distance between Honolulu and O’ahu.
Thought 9: I recall O’ahu is approximately 808 nmi northwest of Honolulu. ←factually wrong
finish[O’ahu]

SIM-CoT

[×] Wake Island

(No tool use) [L×6] [[ANSWER]] <finish> Wake Island </finish>

TAC-CoT (Ours)

[✓] Laysan

Search[Hawaii] → Obs. 1: *... capital and most populous city... Honolulu. ...* [L×3]
Search[808 miles northwest of Honolulu] →
Obs. 2: *Could not find. Similar: [... Laysan (redirect from Laysan Island)...]* [L×3]
Search[Laysan] → Obs. 3: *... Laysan... located 808 nautical miles... northwest of Honolulu. ...* [L×3]
[[ANSWER]] **Laysan**

Figure 9: Case studies on Qwen2.5-0.5B-Instruct. **Gold** Explicit-CoT (SFT) **SIM-CoT** TAC-CoT (ours) . **Red** : error step. (a) Explicit-CoT applies the overtime rate to all 45 h instead of 5 h; SIM-CoT’s fixed latent budget yields a wrong answer with no intermediate steps; TAC-CoT grounds each arithmetic step via calculator calls. (b) SIM-CoT hallucinates “Wake Island” without retrieval; Explicit-CoT searches but rationalizes a wrong answer with a false claim (Thought 9); TAC-CoT spots “Laysan” in search suggestions and retrieves the evidence.

TAC-CoT Inference

(B : segment length, d : hidden size, kv : KV-cache)

(1) Encode prefix [BOS; $q_{1:n}$; [[LATENT]]]

```
# full_ids = [BOS] + question_ids
#           + [LATENT_id]
mask = ones_like(full_ids) # (1, L)
out = model(input_ids=full_ids,
            attention_mask=mask,
            use_cache=True,
            output_hidden_states=True)
kv = out.past_key_values
h = out.hidden_states[-1][:, -1, :]
c = Proj(h.unsqueeze(1)) # (1,1,d)
```

(2) Latent iteration loop ($t = 1, \dots, B$)

```
for t in range(B):
    mask = cat([mask, ones(1,1)],
               dim=-1)
    pos = (mask.cumsum(-1)-1)[:,-1:]
    out = model(
        inputs_embeds = c, # (1,1,d)
        past_key_values = kv,
        attention_mask = mask,
        position_ids = pos,
        use_cache=True,
        output_hidden_states=True)
    h = out.hidden_states[-1][:, -1, :]
    c = Proj(h.unsqueeze(1))
    kv = out.past_key_values
```

(3) Route: action or answer?

```
mask = cat([mask, ones(1,1)], dim=-1)
pos = (mask.cumsum(-1)-1)[:,-1:]
out = model(inputs_embeds=c,
            past_key_values=kv,
            attention_mask=mask,
            position_ids=pos,
            use_cache=True)
kv = out.past_key_values
lg = out.logits[0, -1]
act = max(lg[action_id], lg[ans_id])
ans = lg[answer_id]
```

```
if ans >= act:
    # answer generation (greedy
    # until </finish> or EOS)
else:
    # -> step (4): action generation
```

(4) Action generation and observation injection

```
# greedy generate action tokens
# until </action> stop token

# prepend closing token before obs
inject = [action_close_tok] + obs_ids
mask = cat([mask,
            ones(1,len(inject))], dim=-1)
pos = (mask.cumsum(-1)-1)[:,-len(inject):]
out = model(input_ids=inject,
            past_key_values=kv,
            attention_mask=mask,
            position_ids=pos,
            use_cache=True)
kv = out.past_key_values
# -> back to step (2)
```

Figure 10: Minimal TAC-CoT inference loop. $c_t \in \mathbb{R}^d$ is produced by Proj_θ ($\text{Linear}_{d \rightarrow d_p} \rightarrow \text{GELU} \rightarrow \text{Linear}_{d_p \rightarrow d} \rightarrow \text{LayerNorm}$) and fed back via `inputs_embeds`, bypassing vocabulary projection.

Prompt for QA Tool-Use Trajectory Construction

Instruction and Rules

You are a ReAct agent that answers multi-hop questions using Wikipedia.

Process:

1. Think step by step about what information you need.
2. Use exactly ONE action per turn: Search[entity], Lookup[keyword], or Finish[answer].
3. After your action, you will receive an Observation with the result. Then repeat.

Rules:

- Always write "Thought N:" with your reasoning BEFORE "Action N:".
- Use Search[entity] to find Wikipedia articles.
- Use Lookup[keyword] to find specific info within the current article.
- Use Finish[answer] when you have enough information.
- Only one action per turn.

Few-shot Demonstrations: Examples 1–3

Example 1.

Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?

Thought 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.

Action 1: Search[Colorado orogeny]

Observation 1: The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.

Thought 2: It does not mention the eastern sector. So I need to look up eastern sector.

Action 2: Lookup[eastern sector]

Observation 2: (Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.

Thought 3: The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.

Action 3: Search[High Plains]

Observation 3: High Plains refers to one of two distinct land regions

Thought 4: I need to instead search High Plains (United States).

Action 4: Search[High Plains (United States)]

Observation 4: The High Plains are a subregion of the Great Plains. From east to west, the High Plains rise in elevation

from around 1,800 to 7,000 ft (550 to 2,130 m).[3]

Thought 5: High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.

Action 5: Finish[1,800 to 7,000 ft]

Example 2.

Question: Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who?

Thought 1: The question simplifies to "The Simpsons" character Milhouse is named after who. I only need to search Milhouse and find who it is named after.

Action 1: Search[Milhouse]

Observation 1: Milhouse Mussolini Van Houten is a recurring character in the Fox animated television series The Simpsons voiced by Pamela Hayden and created by Matt Groening.

Thought 2: The paragraph does not tell who Milhouse is named after, maybe I can look up "named after".

Action 2: Lookup[named after]

Observation 2: (Result 1 / 1) Milhouse was named after U.S. president Richard Nixon, whose middle name was Milhous.

Thought 3: Milhouse was named after U.S. president Richard Nixon, so the answer is Richard Nixon.

Action 3: Finish[Richard Nixon]

Example 3.

Question: Which documentary is about Finnish rock groups, Adam Clayton Powell or The Saimaa Gesture?

Thought 1: I need to search Adam Clayton Powell and The Saimaa Gesture, and find which documentary is about Finnish rock groups.

Action 1: Search[Adam Clayton Powell]

Observation 1: Could not find [Adam Clayton Powell]. Similar: ['Adam Clayton Powell III', 'Seventh Avenue (Manhattan)',

'Adam Clayton Powell Jr. State Office Building', 'Isabel Washington Powell', 'Adam Powell', 'Adam Clayton Powell

(film)', 'Giancarlo Esposito'].

Thought 2: To find the documentary, I can search Adam Clayton Powell (film).

Action 2: Search[Adam Clayton Powell (film)]

Observation 2: Adam Clayton Powell is a 1989 American documentary film directed by Richard Kilberg. The film is about the

rise and fall of influential African-American politician Adam Clayton Powell Jr.[3][4] It was later aired as part of

the PBS series The American Experience.

Thought 3: Adam Clayton Powell (film) is a documentary about an African-American politician, not Finnish rock groups. So

the documentary about Finnish rock groups must instead be The Saimaa Gesture.

Action 3: Finish[The Saimaa Gesture]

Figure 11: Prompt used to construct QA tool-use trajectories. We prompt DeepSeek-V3 as a ReAct-style Wikipedia agent with access to Search, Lookup, and Finish.

Prompt for QA Tool-Use Trajectory Construction (continued)

Few-shot Demonstrations: Examples 4–6

Example 4.

Question: What profession does Nicholas Ray and Elia Kazan have in common?

Thought 1: I need to search Nicholas Ray and Elia Kazan, find their professions, then find the profession they have in common.

Action 1: Search[Nicholas Ray]

Observation 1: Nicholas Ray (born Raymond Nicholas Kienzle Jr., August 7, 1911 - June 16, 1979) was an American film

→ director, screenwriter, and actor best known for the 1955 film Rebel Without a Cause.

Thought 2: Professions of Nicholas Ray are director, screenwriter, and actor. I need to search Elia Kazan next and find his

→ professions.

Action 2: Search[Elia Kazan]

Observation 2: Elia Kazan was an American film and theatre director, producer, screenwriter and actor.

Thought 3: Professions of Elia Kazan are director, producer, screenwriter, and actor. So profession Nicholas Ray and Elia

→ Kazan have in common is director, screenwriter, and actor.

Action 3: Finish[director, screenwriter, actor]

Example 5.

Question: Which magazine was started first Arthur's Magazine or First for Women?

Thought 1: I need to search Arthur's Magazine and First for Women, and find which was started first.

Action 1: Search[Arthur's Magazine]

Observation 1: Arthur's Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th

→ century.

Thought 2: Arthur's Magazine was started in 1844. I need to search First for Women next.

Action 2: Search[First for Women]

Observation 2: First for Women is a woman's magazine published by Bauer Media Group in the USA.[1] The magazine was started

→ in 1989.

Thought 3: First for Women was started in 1989. 1844 (Arthur's Magazine) < 1989 (First for Women), so Arthur's Magazine was

→ started first.

Action 3: Finish[Arthur's Magazine]

Example 6.

Question: Were Pavel Urysohn and Leonid Levin known for the same type of work?

Thought 1: I need to search Pavel Urysohn and Leonid Levin, find their types of work, then find if they are the same.

Action 1: Search[Pavel Urysohn]

Observation 1: Pavel Samuilovich Urysohn (February 3, 1898 - August 17, 1924) was a Soviet mathematician who is best known

→ for his contributions in dimension theory.

Thought 2: Pavel Urysohn is a mathematician. I need to search Leonid Levin next and find its type of work.

Action 2: Search[Leonid Levin]

Observation 2: Leonid Anatolievich Levin is a Soviet-American mathematician and computer scientist.

Thought 3: Leonid Levin is a mathematician and computer scientist. So Pavel Urysohn and Leonid Levin have the same type of

→ work.

Action 3: Finish[yes]

Input Query

Question: {question}

Figure 12: Prompt used to construct QA tool-use trajectories, continued from Figure 11. The few-shot demonstrations follow the ReAct prompt format.