DISCARDED BUT NOT FORGOTTEN: ATTENDING TO UNCHOSEN TOKENS FOR LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

In chain-of-thought (CoT) reasoning with large language models (LLMs), each decoding step selects one token to extend the generated reasoning trajectory, while the remaining unchosen tokens are not retained in the generated context. As a result, the states the model would compute by processing these unchosen tokens are never produced and remain inaccessible to later decoding steps, even though they may still contain information useful for subsequent reasoning. To retain this information without generating additional reasoning trajectories, we propose Unchosen Token Memory (UTM), a training-free, inference-time decoding framework that preserves the KV states of top-ranked unchosen tokens as candidate memory. UTM makes these candidate states accessible to subsequent decoding steps through attention, calibrates their influence according to their relative probabilities at the steps where they are produced, and adaptively determines their retention period based on decoding uncertainty. In this way, UTM maintains a single generated reasoning trajectory while allowing subsequent decoding steps to selectively access previously discarded candidate states. Extensive experiments on mathematical reasoning, knowledge-intensive reasoning, and code generation benchmarks further show that UTM achieves higher performance than standard CoT reasoning and recent inference-time reasoning methods across multiple model scales. These results indicate that discarded candidate states provide a useful source of auxiliary information for improving LLM reasoning.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong capabilities across a wide range of complex tasks, including mathematical reasoning, code generation, and knowledge-intensive question answering (Jaech et al., 2024; Guo et al., 2025; Team et al., 2025; Yang et al., 2025). A key factor behind these advances is the emergence of chain-of-thought (CoT) prompting (Wei et al., 2022; Wang et al., 2023), which encourages models to generate intermediate reasoning steps before producing a final answer. By decomposing complex problems into a sequence of simpler reasoning steps, CoT has been shown to significantly improve performance on tasks that require multi-step reasoning. Rather than producing answers in a single step, CoT enables models to construct reasoning trajectories that progressively refine their understanding of the problem. Consequently, the quality of the final answer depends not only on the model's underlying knowledge, but also on how effectively the reasoning trajectory is generated through sequential decoding decisions.

In standard CoT reasoning, each step selects one token and appends it to the generated reasoning trajectory. We refer to the remaining candidates as *unchosen tokens*. Although the next-token distribution already reflects the model's preference over these candidates, the states that the model would compute by actually processing an unchosen token are never produced. They are therefore inaccessible to subsequent decoding steps once the decoding decision is made, as illustrated in Fig. 1(a). Meanwhile, recent studies suggest that internal states produced during failed reasoning trajectories may still contain recoverable task-relevant information, and that reasoning failures are often concentrated at a sparse set of high-uncertainty decision points (Mehrafarin et al., 2026; Shen et al., 2026). Moreover, branching on top-ranked alternatives instead of the greedy token can reveal correct reasoning paths that greedy decoding misses (Wang & Zhou, 2024). These findings motivate us to consider whether information associated with plausible unchosen tokens can be retained for subsequent reasoning.

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

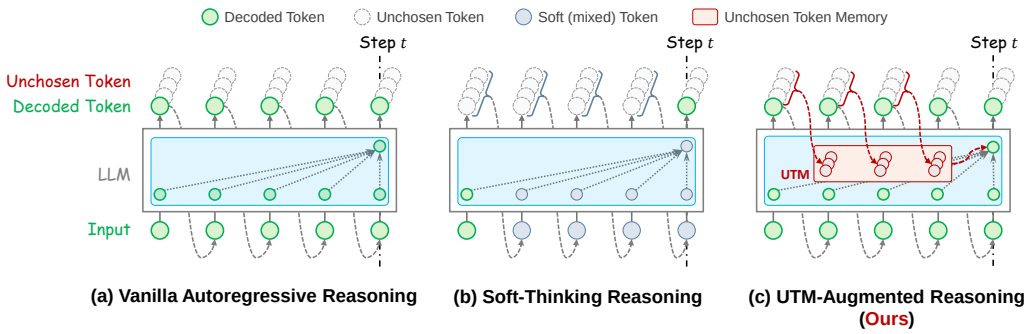


Figure 1: **Comparison of vanilla autoregressive reasoning, Soft-Thinking, and UTM.** Vanilla decoding discards unchosen candidates, Soft-Thinking merges candidate information into a single soft representation, whereas UTM stores unchosen-token KV states as memory slots for later steps.

Recent inference-time decoding methods make broader use of information beyond the selected token. Multi-trajectory methods such as self-consistency Wang et al. (2023) and beam search explore alternatives explicitly, but at the cost of generating additional reasoning trajectories. Soft-Thinking (Zhang et al., 2026) instead combines candidate-token embeddings into a single probability-weighted soft representation as shown in Fig. 1(b), while its stochastic variant uses Gumbel-Softmax to enhance exploration during decoding (Wu et al., 2025). However, these soft-representation methods merge candidate information into a single representation at the current step, rather than constructing and preserving separate KV states for the unchosen tokens, so later steps cannot selectively consult individual alternatives. This leaves open how such KV states can be retained and reused across decoding steps without generating additional reasoning trajectories.

To address this limitation, we propose **Unchosen Token Memory** (UTM), an inference-time decoding framework that preserves the KV states of top-ranked unchosen tokens as auxiliary memory, as illustrated in Fig. 1(c). UTM follows a single reasoning trajectory while maintaining candidate memory separately from the standard prefix KV cache. When candidate KV storage is activated, UTM stores one-step KV states for top-ranked unchosen tokens without appending their tokens to the visible reasoning trajectory or decoding them into additional continuations. These candidate KV states remain accessible to subsequent decoding steps through the attention context. Since each candidate KV state is computed by the same model, prefix, and position as the selected token, UTM can be applied to off-the-shelf LLMs without training. However, naively attending to all candidate states can hurt performance. UTM therefore stores candidates only at high-uncertainty steps, calibrates each by its relative probability, and adaptively limits how long it remains active. Our contributions are summarized as follows.

- First, we identify candidate retention as an overlooked opportunity in single-trajectory CoT reasoning: plausible unchosen-token states are discarded after each decoding decision even though they may remain useful for later reasoning.
- Second, we propose **Unchosen Token Memory**, which preserves the KV states of top-ranked unchosen tokens as auxiliary memory and makes them accessible to subsequent decoding steps through attention. UTM uses candidate KV storage, adaptive memory retention, and memory-augmented decoding to manage and utilize candidate memory.
- Third, we conduct extensive experiments across mathematical reasoning, knowledge-intensive reasoning, and code generation benchmarks, demonstrating the effectiveness and generalizability of UTM and systematically analyzing its candidate selection, memory representation, adaptive mechanisms, and inference efficiency.

2 Related Work

2.1 Reasoning with Large Language Models

Chain-of-thought (CoT) reasoning has become a central approach for improving the reasoning ability of LLMs. By generating intermediate reasoning steps before the final answer, CoT decomposes

complex problems into simpler subproblems and improves performance on mathematical, symbolic, and commonsense reasoning tasks (Wei et al., 2022; Kojima et al., 2022; Hao et al., 2024). Building on CoT, self-consistency samples multiple reasoning trajectories and aggregates their answers to improve robustness (Wang et al., 2023), while tree-of-thought and related search-based methods organize intermediate thoughts into structured search spaces for exploring different reasoning directions (Yao et al., 2023a; Besta et al., 2024; Yao et al., 2023b). These methods enhance reasoning mainly by generating or searching over multiple explicit reasoning trajectories. In contrast, our work focuses on information already present within single-trajectory generation: unchosen tokens that are considered by the model but not selected into the generated reasoning trajectory. Rather than expanding these candidates into separate trajectories, we preserve their associated states and make them accessible to subsequent decoding steps.

2.2 Inference-Time Decoding for Reasoning

Reasoning performance can be improved by better utilizing information available during decoding beyond the selected token. Standard CoT reasoning relies on greedy decoding or sampling, where one token is selected from the next-token distribution at each step to construct a single reasoning trajectory. Recent methods aim to improve decoding by making use of information beyond the selected token (Zhang et al., 2026; Wu et al., 2025; Kuzina et al., 2026; Mehrafarin et al., 2026; Zhu et al., 2025; He et al., 2026; Tan et al., 2026b; Zhuang et al., 2026; Fu & Luo, 2026; Tan et al., 2026a; Zeyu et al., 2026; Jain & Rappazzo, 2025). Soft-Thinking replaces hard token selection with continuous concept tokens constructed from probability-weighted token embeddings, allowing information from multiple unchosen tokens to be aggregated into a soft representation (Zhang et al., 2026). Soft-Thinking-Gumbel further introduces stochasticity into this representation construction (Wu et al., 2025). LED follows a different route within hard decoding by leveraging intermediate-layer posteriors to adjust token selection when the final-layer distribution becomes overly confident (Tan et al., 2026b). These methods show that useful information beyond the selected token can be leveraged during inference-time decoding. In this paper, we preserve the KV states of unchosen tokens and make them accessible to later decoding steps through the attention context. This allows the model to reuse information associated with candidates that would otherwise become inaccessible while generating a single reasoning trajectory.

3 Method

3.1 Overview

We propose **Unchosen Token Memory** (UTM), which augments standard CoT reasoning with an auxiliary KV memory for unchosen tokens. Instead of generating additional reasoning trajectories, UTM continues to follow a single CoT trajectory while preserving the KV states of top-ranked unchosen tokens. These candidate KV states are stored as additional key-value entries and can be accessed by subsequent decoding steps through attention. In this way, UTM enables the model to reuse candidate information that would otherwise become inaccessible while maintaining the standard autoregressive generation process.

At each decoding step, UTM computes the uncertainty of the next-token distribution and uses two independent entropy-based criteria to control candidate KV storage and memory-augmented decoding. The storage criterion δ_{sto} determines whether KV states are stored for top-ranked unchosen tokens at the current step, while the decoding criterion δ_{dec} determines whether active candidate memory is included in the attention context of the subsequent decoding forward. Newly stored candidate KV states are stored for subsequent decoding steps, and adaptive memory retention determines how long they remain active. Candidate memory is updated online during generation, where states that exceed their retention periods are removed rather than accumulated over the entire reasoning trajectory.

3.2 Candidate KV Storage

At decoding step t, let $y_{<t} = (y_1, \ldots, y_{t-1})$ denote the generated prefix. Given this prefix, the current decoding forward produces a next-token distribution:

$$p_t = P(\cdot \mid y_{<t}). \tag{1}$$

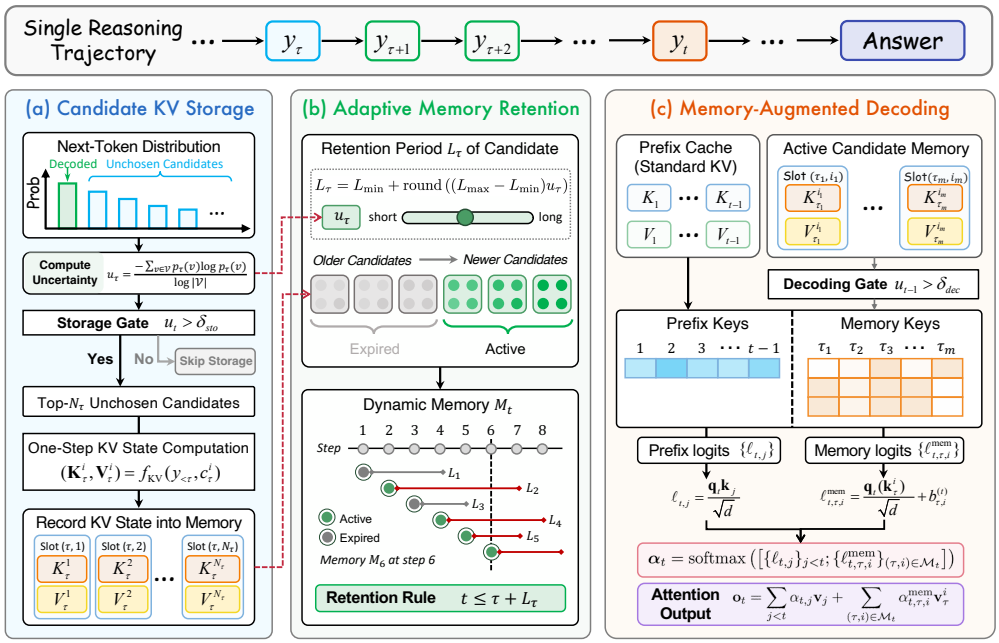


Figure 2: Overview of Unchosen Token Memory (UTM).

Here, p_t denotes the full-vocabulary distribution before applying top-k, top-p, and min-p filtering. We denote by $\tilde{p}_t$ the filtered and renormalized distribution used by the base LLM's decoding strategy for token selection. We measure the uncertainty of the current decoding step using normalized entropy over the full vocabulary:

$$u_t = \frac{-\sum_{v \in \mathcal{V}} p_t(v) \log p_t(v)}{\log |\mathcal{V}|}, \tag{2}$$

where $\mathcal{V}$ denotes the model vocabulary. The resulting uncertainty score satisfies $u_t \in [0, 1]$. UTM does not store candidate KV states at every decoding step. Instead, candidate KV storage is activated only when the uncertainty score u_t exceeds a predefined threshold δ_{sto}. When the threshold is not reached, the decoding forward pass continues without adding new candidate memory.

The base LLM selects the next token y_t from $\tilde{p}_t$ according to its decoding strategy. The selected token is appended to the reasoning trajectory, and the standard prefix KV cache is updated in the same way as standard CoT reasoning. When candidate KV storage is activated, UTM reuses $\tilde{p}_t$ to identify top-ranked unchosen tokens. Specifically, after excluding the selected token y_t, we retain

$$\mathcal{C}_t = \{c_t^1, c_t^2, \ldots, c_t^{N_t}\}, \qquad c_t^i \neq y_t, \tag{3}$$

where N_t is the number of retained unchosen tokens and c_t^i denotes the i-th candidate at step t. For each candidate $c_t^i \in \mathcal{C}_t$, we materialize its corresponding one-step KV state by forwarding the unchosen token under the same generated prefix:

$$(\mathbf{K}_t^i, \mathbf{V}_t^i) = f_{\text{KV}}(y_{<t}, c_t^i). \tag{4}$$

Here, $f_{\text{KV}}(\cdot)$ denotes the model forward operation that returns the key and value states produced by appending the unchosen token to the current prefix. This operation retains only the KV state associated with the unchosen token. It does not append the candidate to the visible reasoning trajectory, modify the standard prefix KV cache, or decode the candidate into a separate continuation.

The newly stored candidate KV states are denoted by

$$\mathcal{S}_t = \{(\mathbf{K}_t^i, \mathbf{V}_t^i)\}_{i=1}^{N_t}. \tag{5}$$

These states are added to candidate memory for subsequent decoding steps, while their periods are controlled by the adaptive memory retention policy described next.

3.3 Adaptive Memory Retention

Maintaining candidate KV states introduces additional memory overhead. Since candidate information generated at different decoding steps may remain useful for different durations, UTM does not retain all candidate states throughout generation. Instead, it adaptively controls the period of each candidate memory slot according to the uncertainty of the decoding step where the slot is stored.

For candidate states stored at step τ, we map the uncertainty score u_τ to a retention period:

$$L_\tau = L_{\min} + \operatorname{round}\left((L_{\max} - L_{\min})u_\tau\right). \tag{6}$$

where $L_{\min}$ and $L_{\max}$ denote the minimum and maximum retention periods. A larger uncertainty score therefore assigns a longer retention period, while candidate states stored at more confident decoding steps are removed earlier.

A candidate memory slot stored at step τ remains active at step t if

$$t - \tau \le L_\tau. \tag{7}$$

Let $\mathcal{M}_t$ denote the set of active candidate memory slots available at step t. During generation, $\mathcal{M}_t$ is maintained under a fixed memory budget. When the number of active candidate slots exceeds the budget, the oldest slots are removed first. This adaptive retention policy limits the growth of candidate memory while keeping candidate states from more uncertain decoding steps accessible for longer.

3.4 Memory-Augmented Decoding

Given the active candidate memory $\mathcal{M}_t$, UTM determines whether to include it in the attention context at decoding step t using an independent entropy-based criterion. Specifically, candidate memory is used at step t only when the uncertainty score u_{t-1} exceeds a predefined decoding threshold. Otherwise, or when $\mathcal{M}_t$ is empty, UTM reduces to standard attention over the generated prefix.

Let $(\mathbf{K}_{<t}, \mathbf{V}_{<t})$ denote the standard KV cache of the generated prefix $y_{<t}$, where $\mathbf{k}_j$ and $\mathbf{v}_j$ are the key and value vectors of the j-th generated token. Let $\mathbf{q}_t$ denote the query vector at decoding step t. For each generated token $j < t$, the standard attention logit is

$$\ell_{t,j} = \frac{\mathbf{q}_t \mathbf{k}_j^\top}{\sqrt{d}}, \tag{8}$$

where d is the attention hidden dimension. When memory-augmented decoding is activated, UTM keeps the attention logits over the generated prefix unchanged and augments them with candidate memory logits. Suppose a candidate KV state $(\mathbf{K}_\tau^i, \mathbf{V}_\tau^i)$ is stored from an unchosen token c_τ^i at an earlier decoding step $\tau < t$. For simplicity, we denote its key and value vectors by $(\mathbf{k}_\tau^i, \mathbf{v}_\tau^i)$.

Since the selected token at step τ is sampled from the decoding distribution, it is not necessarily the most probable token. We therefore calibrate each candidate memory slot relative to the top-ranked token. Let $\tilde{p}_\tau$ denote the distribution used by the base LLM's decoding strategy for selection, and let r_τ denote its top-ranked token. The attention logit of candidate memory slot (τ, i) is defined as

$$\ell_{t,\tau,i}^{\mathrm{mem}} = \frac{\mathbf{q}_t (\mathbf{k}_\tau^i)^\top}{\sqrt{d}} + b_{\tau,i}, \tag{9}$$

where the relative prior is

$$b_{\tau,i} = \gamma \left(\log \tilde{p}_\tau(c_\tau^i) - \log \tilde{p}_\tau(r_\tau)\right), \tag{10}$$

and γ is a fixed calibration scale.

The query-key term measures the relevance of the candidate memory slot to the decoding step, while $b_{\tau,i}$ calibrates the slot according to its relative probability at the step where it is stored. Using the top-ranked token as the reference prevents the prior from depending on the randomness of the sampled token. Under this reference, the top-ranked token receives zero bias, while other candidates are calibrated according to their probability gaps. The relative prior remains fixed while the candidate memory slot is active, whereas its period is controlled by adaptive memory retention.

We compute a single attention distribution over generated-prefix tokens and active candidate memory slots:

$$\boldsymbol{\alpha}_t = \operatorname{softmax}\left(\left[\{\ell_{t,j}\}_{j<t}; \{\ell_{t,\tau,i}^{\mathrm{mem}}\}_{(\tau,i)\in\mathcal{M}_t}\right]\right). \tag{11}$$

We denote the resulting attention weights assigned to prefix token j and candidate memory slot (τ, i) by $\alpha_{t,j}$ and $\alpha^{\text{mem}}_{t,\tau,i}$, respectively. The attention output is

$$\mathbf{o}_t = \sum_{j<t} \alpha_{t,j}\mathbf{v}_j + \sum_{(\tau,i)\in\mathcal{M}_t} \alpha^{\text{mem}}_{t,\tau,i}\mathbf{v}^i_\tau. \tag{12}$$

This formulation preserves the standard attention over the generated prefix while adding calibrated candidate KV slots as auxiliary memory.

4 Experiment

4.1 Datasets and Evaluation Metrics

We evaluate our method on nine reasoning benchmarks. The mathematical reasoning benchmarks include GSM8K(Cobbe et al., 2021), MATH-500(Hendrycks et al., 2021), AMC23 (AMC, 2025), AIME2024, and AIME2025 (AMC, 2025), covering problems from grade-school arithmetic to challenging competition-level reasoning. The knowledge-intensive reasoning benchmark is GPQA-Diamond (Rein et al., 2023), which requires expert-level scientific knowledge. The code generation benchmarks include HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2025), which evaluate program synthesis with unit tests.

We report accuracy for all benchmarks. For mathematical and knowledge-intensive reasoning tasks, correctness is determined by matching the extracted final answer against the ground truth under the benchmark-specific evaluation rule. For code generation tasks, correctness is determined by whether the program passes the official unit tests. To ensure a fair comparison, all methods use the same prompt format, answer extraction procedure, and evaluation protocol.

4.2 Baselines

We compare our method with standard CoT reasoning and recent decoding-time reasoning methods, such as **CoT-Greedy** (Wei et al., 2022), **CoT-Sampling** (Wei et al., 2022), **Soft-Thinking** (Zhang et al., 2026) and **Soft-Thinking-Gumbel** (Wu et al., 2025). Our method follows the same single-trajectory decoding setting as CoT-Sampling, but augments it with candidate KV memory. Instead of replacing the selected token with a soft representation or generating multiple complete reasoning trajectories, our method preserves top-ranked unchosen-token states as additional KV memory slots that can be accessed by later decoding steps.

4.3 Main Results

Table 1 presents results across four model configurations and two model families, including three Qwen3 scales and DeepSeek-R1-Distill-Llama-8B. Our method achieves the best macro-average performance across four model configurations, suggesting that candidate KV memory provides consistent gains over standard CoT reasoning. Unlike Soft-Thinking, which mixes candidate information in the embedding space, our method keeps the trajectory discrete and allows later steps to selectively access candidate states through attention. This may explain why the gains are more stable across both reasoning and code-generation benchmarks.

The improvements are not uniform across datasets. Gains are most evident on GPQA-Diamond and several competition-level math benchmarks. These tasks involve multiple plausible intermediate directions, making them more suitable for candidate-memory reuse. In contrast, on highly saturated datasets such as GSM8K and MATH-500, the margin between methods is smaller because the base LLM solves most samples. We observe that the benefit does not simply increase with model scale. The 8B model shows the most consistent improvement, while the 32B model has stronger baseline reasoning ability and leaves a smaller performance margin for inference-time improvements.

4.4 Ablation Study

We perform ablation studies to examine how the key memory mechanisms in UTM contribute to its effectiveness. We consider two aspects: how candidate memory is calibrated and retained, and when it is stored and accessed during decoding. More details are provided in Appendix A.2.

Table 1: Main results on mathematical reasoning, knowledge-intensive reasoning, and code generation benchmarks. Accuracy is reported in percentage. Subscripts denote standard deviations over three runs. AVG. denotes the macro-average of the mean accuracies across all benchmarks. The best results are in bold, and the second-best are underlined.

Method	Math Reasoning					Knowledge	Code Generation			Overall
	AIME24	AIME25	AMC23	GSM8K	MATH-500	GPQA	HumanEval	LCB	MBPP	AVG.
Qwen3-4B										
CoT-Sampling	72.2±3.9	62.2±3.9	**98.3±1.4**	95.0±0.5	95.9±0.6	52.7±0.6	93.3±1.6	**56.8±3.0**	93.5±0.6	80.0
CoT-Greedy	64.4±1.9	57.8±1.9	96.7±1.4	94.0±0.3	95.1±0.4	48.7±0.6	85.8±0.9	45.6±0.7	86.8±0.5	75.0
Soft-Thinking	67.8±5.1	64.4±6.9	90.8±3.8	94.7±0.5	96.2±0.7	49.8±1.6	88.2±1.4	49.0±1.8	89.0±1.0	76.7
Gumbel	**78.9±3.8**	67.8±5.1	95.8±2.9	95.0±0.4	96.5±0.5	51.4±1.2	**93.8±0.8**	55.0±1.1	92.9±0.7	80.8
Ours	77.8±1.9	**71.1±3.3**	**98.3±1.4**	**95.9±0.2**	**96.9±0.3**	**52.9±0.9**	93.5±0.6	56.2±0.8	**93.9±0.5**	**81.8**
Qwen3-8B										
CoT-Sampling	74.4±1.9	68.9±5.1	**96.7±3.8**	95.4±0.2	96.6±0.8	57.1±1.8	93.9±0.6	60.7±1.5	94.0±0.4	82.0
CoT-Greedy	74.4±1.9	54.4±3.3	95.8±1.4	95.7±0.2	96.5±0.2	52.8±0.7	87.5±0.6	50.5±0.5	89.4±0.4	77.4
Soft-Thinking	72.2±6.9	67.8±5.1	91.7±3.8	**96.1±0.3**	96.2±0.5	53.4±1.4	93.0±1.5	53.2±1.7	91.0±1.2	79.3
Gumbel	74.4±3.3	67.8±3.8	93.3±2.9	95.8±0.4	**97.7±0.4**	58.4±1.4	92.4±1.1	56.7±1.0	94.5±0.8	81.2
Ours	**76.7±3.3**	**75.6±1.9**	95.8±1.4	95.9±0.1	97.5±0.3	**59.9±0.8**	**94.0±0.7**	**60.8±0.6**	**96.1±0.5**	**83.6**
DeepSeek-R1-Distill-Llama-8B										
CoT-Sampling	41.1±4.2	33.3±2.7	87.5±2.0	78.0±0.7	**85.7±0.7**	44.9±0.2	90.8±0.6	38.4±1.2	**89.2±0.9**	65.4
CoT-Greedy	37.8±3.2	**34.4±1.6**	70.8±1.2	**78.3±0.8**	82.5±0.7	42.1±0.5	78.8±1.1	28.1±1.6	76.8±1.6	58.8
Soft-Thinking	36.7±2.7	28.9±1.6	72.5±2.0	77.3±1.0	84.2±1.0	41.0±0.6	85.3±1.0	32.5±0.7	78.9±1.4	59.7
Gumbel	42.2±1.6	31.1±1.6	79.2±3.1	76.4±0.4	83.2±1.0	42.9±1.4	86.6±1.0	41.4±1.2	82.6±3.2	62.8
Ours	**52.2±1.6**	33.3±2.7	**91.7±4.2**	77.2±0.7	85.3±0.7	**46.0±0.2**	**91.3±0.8**	**42.8±0.2**	88.7±0.7	**67.6**
Qwen3-32B										
CoT-Sampling	**84.4±5.1**	66.7±3.3	96.7±2.9	96.2±0.4	97.7±0.2	**66.3±1.5**	97.5±1.2	65.8±0.6	96.9±0.4	85.4
CoT-Greedy	77.8±1.9	74.4±3.3	**99.2±1.4**	96.0±0.2	96.7±0.2	60.9±0.6	97.2±0.5	**66.5±0.4**	97.0±0.3	85.1
Soft-Thinking	82.2±5.1	64.4±6.9	97.5±2.5	96.1±0.4	**98.3±0.6**	61.9±1.5	97.2±1.2	61.2±1.4	96.3±0.9	83.9
Gumbel	81.1±3.8	74.4±3.3	**99.2±1.4**	96.0±0.3	97.4±0.4	62.9±1.0	**97.8±0.8**	64.2±0.9	96.4±0.6	85.5
Ours	81.1±3.8	**80.0±3.3**	98.3±1.4	**96.9±0.2**	**98.3±0.2**	66.0±0.7	**97.8±0.6**	65.0±0.5	**97.2±0.4**	**86.7**

Table 2: Incremental ablation of UTM on Qwen3-8B. Components are added cumulatively from top to bottom. CM: candidate memory; RP: relative prior; AR: adaptive retention; Gate: storage and decoding gates.

Method	Components				Math Reasoning					Knowledge.	Code Generation			Avg.
	CM	RP	AR	Gate	AIME24	AIME25	AMC23	GSM8K	MATH	GPQA	HEval	LCB	MBPP	
CoT					74.4	68.9	**96.7**	95.4	96.6	57.1	93.9	60.7	94.0	82.0
	✓				70.0	63.3	90.0	95.5	96.8	58.1	**95.1**	59.9	95.7	80.5
	✓	✓			76.7	73.3	90.0	**95.9**	95.9	59.1	93.3	57.7	95.7	82.1
	✓	✓	✓		76.7	73.3	95.0	95.8	96.2	59.5	93.2	58.8	95.5	82.7
UTM	✓	✓	✓	✓	**76.7**	**75.6**	95.8	**95.9**	**97.5**	**59.9**	94.0	**60.8**	**96.1**	**83.6**

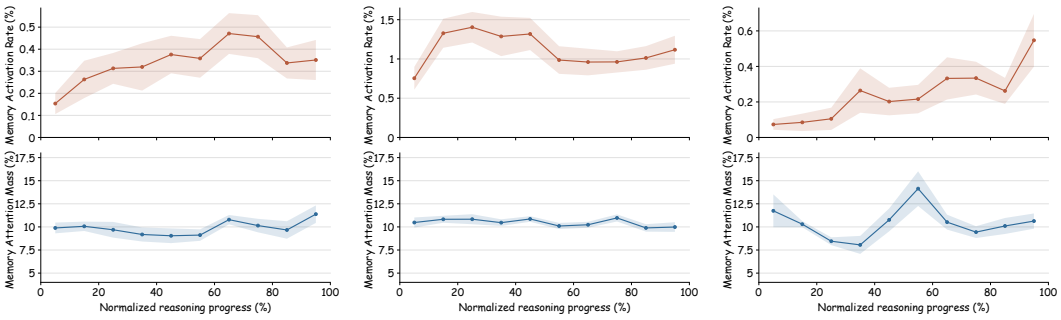
Incremental Ablation of UTM Components. Starting from standard CoT-Sampling, we progressively add candidate memory, the relative prior, adaptive retention, and the storage and decoding gates, as shown in Table 2. Candidate memory preserves unchosen-token states for later reasoning, the relative prior calibrates their attention contribution according to their probability when generated, adaptive retention controls how long they remain active based on decoding uncertainty, and the storage and decoding gates determine when candidate memory is created and accessed.

Adding candidate memory alone decreases the average accuracy from 82.0 to 80.5, showing that directly reusing candidate KV states without additional control can introduce distracting alternatives. Adding the relative prior recovers the average accuracy to 82.1, while adaptive retention further improves it to 82.7. Finally, introducing the storage and decoding gates raises the average accuracy to 83.6. These results show that candidate memory becomes effective when its influence, lifetime, and usage are properly controlled.

Effect of the Gated Storage and Decoding. We further study whether candidate memory should be stored and accessed at every decoding step. Storing candidate KV states at all steps introduces additional forward computation, while accessing active memory at all steps enlarges the attention

Table 3: Effect of the storage and decoding gates on Qwen3-8B. *Always*: storing or using candidate memory at every step; *Gated*: triggering it only when the uncertainty exceeds the threshold. Acc.: accuracy (%); Time: latency (s/sample). The last row corresponds to the full UTM.

Storage	Decoding	MATH-500		GPQA		HumanEval		Avg.	
		Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓
Always	Always	96.2	168.3	59.5	139.3	93.2	57.9	83.0	121.8
Gated	Always	95.4	85.3	57.6	104.2	93.9	32.0	82.3	73.8
Always	Gated	97.2	81.9	58.7	105.7	**94.4**	79.8	83.4	89.1
Gated	Gated	**97.5**	**33.2**	**59.9**	**65.5**	94.0	**25.2**	**83.8**	**41.3**



(a) MATH-500 (b) GPQA-Diamond (c) HumanEval

Figure 3: Candidate memory usage over normalized reasoning progress. The top panels report the memory activation rate, i.e., the percentage of decoding steps with available candidate memory, while the bottom panels report the attention mass allocated to candidate memory when it is active. Shaded regions indicate the standard error of the mean.

context and may expose the model to unnecessary alternatives. Table 3 therefore independently varies storage and decoding between "Always" and "Gated". Gated storage determines when new candidate KV states are created, whereas gated decoding determines when active candidate memory is included in the attention context.

Applying both storage and decoding at all steps incurs the highest latency. Gating storage substantially reduces latency with only a small drop in average accuracy, while gating decoding improves both accuracy and efficiency. Combining gated storage and gated decoding achieves the best overall accuracy with the lowest latency. These results show that UTM benefits from selectively controlling not only what candidate information is retained, but also when it is created and used.

4.5 Analysis

Effect of Candidate Memory Usage We further analyze how candidate memory is utilized during decoding. Since storing candidate KV states introduces additional computation and memory overhead, UTM does not activate candidate memory at every decoding step. Instead, the adaptive memory mechanism selectively maintains candidate states when additional reasoning alternatives are likely to be useful. For each reasoning step, we measure two quantities: the memory activation rate, which indicates how frequently candidate memory is available, and the attention mass assigned to candidate memory when activated, which measures how much the model relies on these states.

As shown in Figure 3, candidate memory is activated only at a small fraction of reasoning steps across different benchmarks. However, once activated, candidate memory receives attention from the current decoding step, demonstrating that the retained candidate states provide useful information when they are available. This sparse activation pattern reflects the design trade-off of UTM: instead of continuously expanding the decoding context with candidate states, UTM selectively introduces candidate memory to reduce unnecessary computation while preserving useful alternative reasoning signals.

Table 4: Accuracy and relative latency comparison on Qwen3-8B across three benchmarks. Relative latency is normalized by Chain-of-Thought on each benchmark. #Trajectories denotes the number of decoding trajectories. Best-of-N and Self-Consistency sample N independent trajectories, selecting the final response by sequence likelihood and majority voting, respectively. Beam Search maintains B interacting trajectories that are pruned at each decoding step.

		MATH-500		GPQA		HumanEval	
Method	**#Trajectories**	**Acc.↑**	**Rel. Latency↓**	**Acc.↑**	**Rel. Latency↓**	**Acc.↑**	**Rel. Latency↓**
CoT-Sampling	1	96.6	1.00×	57.1	1.00×	93.9	1.00×
Best-of-N (N=4)	4	96.6	2.16×	59.1	2.01×	93.9	2.50×
Self-Consistency (N=4)	4	**97.6**	2.47×	57.8	2.03×	**94.0**	2.58×
Beam Search (B=2)	2	97.2	7.84×	57.9	8.35×	84.1	10.35×
UTM (Ours)	1	97.5	1.30×	**59.9**	1.32×	**94.0**	1.42×

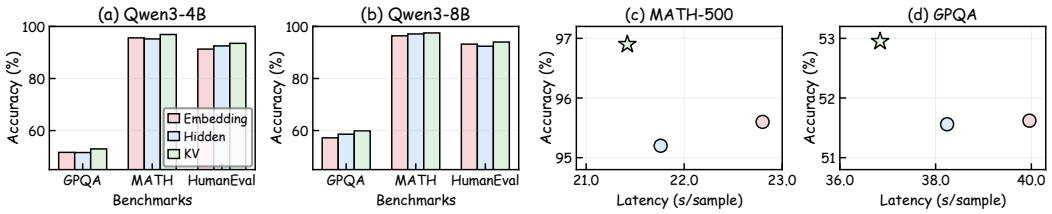


Figure 4: Effect of candidate memory representation. (a,b) Accuracy of embedding, hidden-state, and KV representations across models. (c,d) Accuracy–latency trade-off on Qwen3-4B for MATH-500 and GPQA.

Effect of Inference-Time Decoding Strategies Table 4 highlights how different decoding strategies trade computation for reasoning performance. Self-Consistency (N=4) slightly outperforms UTM on MATH-500, but its advantage does not extend to GPQA and comes with substantially higher latency; Best-of-N (N=4) incurs similar overhead because both methods generate four complete trajectories before selection or voting. Beam Search (B=2) is considerably more expensive because it maintains and expands up to two active prefixes during decoding, yet its accuracy remains inconsistent across benchmarks. In contrast, UTM stores KV states for unchosen tokens and selectively reuses them within a single generated trajectory, achieving the best result on GPQA, tying for the best result on HumanEval, and remaining competitive on MATH-500 with a much smaller latency increase.

Effect of Candidate Memory Representation We compare embeddings, hidden states, and KV states as the memory representation, while keeping candidate selection, retention, and decoding settings unchanged. As shown in Figure 4(a,b), KV memory achieves the best accuracy across MATH-500, GPQA, and HumanEval on both Qwen3-4B and Qwen3-8B. This suggests that preserving discarded candidates in their attention-ready KV form is more effective than storing token-level or generic hidden representations. Figure 4(c,d) compares the accuracy–latency trade-off on MATH-500 and GPQA. KV memory obtains the highest accuracy with the lowest latency on both datasets, since its states can be directly incorporated into the attention context without projection during decoding. These results support KV states as an effective and efficient memory unit for UTM.

5 Conclusion

We introduced Unchosen Token Memory (UTM), an inference-time framework that preserves the KV states of top-ranked unchosen tokens as auxiliary memory for subsequent reasoning. UTM maintains a single generated reasoning trajectory, while storage, adaptive retention, and memory-augmented decoding control when candidate states are created, kept, and accessed. Experiments across multiple benchmarks show consistent improvements over standard CoT and recent inference-time reasoning methods. These results demonstrate that the unchosen token state during autoregressive decoding can remain useful for later reasoning.

AI USE STATEMENT

In this work, we used generative AI tools for feedback on research methodology and experimental design, analysis of experimental results, code editing and debugging, and manuscript editing. All AI-assisted outputs were reviewed and, where applicable, verified by the authors before being incorporated into the work. We take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Section 3 describes the proposed method, and Section 4 provides the experimental setup, evaluation protocol, ablation studies, and analyses. Sections A and B provide additional implementation details, hyperparameter settings, and supplementary experimental results needed for reproduction.

REFERENCES

AMC. American invitational mathematics examination. https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

Renyu Fu and Guibo Luo. Selar: Selective latent reasoning in large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7073–7087, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Kaifeng He, Mingwei Liu, Chong Wang, Zike Li, Yanlin Wang, Xin Peng, and Zibin Zheng. Adadec: An uncertainty-guided lookahead decoding framework for llm-based code generation. *Proceedings of the ACM on Software Engineering*, 3(FSE):3047–3069, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Adit Jain and Brendan Rappazzo. Learning to reason with mixture of tokens. *arXiv preprint arXiv:2509.21482*, 2025.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, pp. 58791–58831, 2025.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213, 2022.

Anna Kuzina, Maciej Pioro, Paul N Whatmough, and Babak Ehteshami Bejnordi. Kava: Latent reasoning via compressed kv-cache distillation. In *The Fourteenth International Conference on Learning Representations*, 2026.

Houman Mehrafarin, Amit Parekh, and Ioannis Konstas. When chain-of-thought fails, the solution hides in the hidden states. *arXiv preprint arXiv:2604.23351*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Changshuo Shen, Leheng Sheng, Yuxin Chen, Xiang Wang, and An Zhang. Reasoning can be restored by correcting a few decision tokens. In *Forty-third International Conference on Machine Learning*, 2026.

Wenhui Tan, Minghao Li, Xiaoqian Ma, Siqi Fan, Xiusheng Huang, Liujie Zhang, Ruihua Song, and Weihang Chen. Pair-in, pair-out: Latent multi-token prediction for efficient llms. *arXiv preprint arXiv:2605.27255*, 2026a.

Wenhui Tan, Fiorenzo Parascandolo, Enver Sangineto, Jianzhong Ju, Zhenbo Luo, Qian Cao, Rita Cucchiara, Ruihua Song, and Jian Luan. Restoring exploration after post-training: Latent exploration decoding for large reasoning models. *arXiv preprint arXiv:2602.01698*, 2026b.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. *Advances in Neural Information Processing Systems*, 37:66383–66409, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Junhong Wu, Jinliang Lu, Zixuan Ren, Gangqiang Hu, Zhi Wu, Dai Dai, and Hua Wu. Llms are single-threaded reasoners: Demystifying the working mechanism of soft thinking. *arXiv preprint arXiv:2508.03440*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Ilya Shafran, Thomas Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

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

XING Zeyu, Xing Li, Hui-Ling Zhen, Mingxuan Yuan, and Sinno Jialin Pan. Beyond speedup-utilizing kv cache for sampling and reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. In *Advances in Neural Information Processing Systems*, volume 38, pp. 168990–169012, 2026.

Qinglin Zhu, Runcong Zhao, Hanqi Yan, Yulan He, Yudong Chen, and Lin Gui. Soft reasoning: Navigating solution spaces in large language models through controlled embedding exploration. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. Mixture of inputs: Text generation beyond discrete token sampling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 26363–26387, 2026.

A Implementation Details

A.1 Experimental Implementation Details

We evaluate UTM on Qwen3-4B, Qwen3-8B, Qwen3-32B, and DeepSeek-R1-Distill-Llama-8B without additional training or parameter updates. Generation uses a temperature of 0.6, Top-p of 0.95, Top-k of 30, Min-p of 0.001, and a maximum generation length of 32,768 tokens.

When candidate KV construction is activated, UTM retains up to eight highest-probability unchosen tokens. We set the relative-prior scale to $\gamma = 0.5$, the minimum and maximum retention horizons to $L_{\min} = 4$ and $L_{\max} = 64$, respectively, and the maximum candidate-memory capacity to 512 slots. Candidate rollout and candidate-branch logit mixing are not used. UTM is active only during the thinking phase, while final-answer generation follows the standard decoding process.

We evaluate mathematical reasoning on AIME2024, AIME2025, AMC23, GSM8K, and MATH-500; knowledge-intensive reasoning on GPQA-Diamond; and code generation on HumanEval, MBPP, and LiveCodeBench. Each experiment is conducted using three random seeds: 0, 1, and 2. For each seed, we generate one response per problem, and report the mean and standard deviation over the three complete evaluation runs. Mathematical and knowledge-intensive reasoning tasks use benchmark-specific answer extraction and matching rules, while code-generation outputs are evaluated using the official unit tests.

For a fair comparison, we use a single fixed hyperparameter configuration for each method across all benchmarks, without dataset-specific tuning. Shared decoding parameters are kept consistent across UTM and the baselines whenever applicable, while method-specific parameters are also fixed across datasets. All baseline results are reproduced under these settings rather than selected from configurations individually tuned for particular benchmarks. All experiments are conducted on four NVIDIA A800 GPUs, and the reported latency comparisons are measured under the same hardware setting.

A.2 Additional Implementation Details

The resulting uncertainty score lies in $[0, 1]$. We use a fixed threshold of $\delta_{\text{sto}} = 0.30$ and $\delta_{\text{dec}} = 0.30$ for both candidate KV construction and memory-augmented decoding. The uncertainty computed from the current main-path forward determines whether candidate states are constructed at the current step and whether active candidate memory is included in the subsequent decoding forward.

Candidate KV states are stored separately from the standard prefix KV cache at every transformer layer. A candidate constructed at step τ uses the same position ID as the visible token generated at that step. RoPE is applied during the candidate forward before cache insertion, and the stored candidate keys are reused directly during subsequent decoding without additional rotation. Active candidate-memory slots are accessible only to subsequent decoding queries through the augmented attention context. They do not enter the visible token sequence, advance the main-path position, or alter the causal relations among generated-prefix tokens.

For the naive candidate-memory variant in Table 2, the same unchosen-token candidates are materialized as one-step KV states under the generated prefix and inserted as auxiliary memory slots for subsequent attention. The relative-prior bias is removed, so candidate-memory logits rely only on standard query–key similarity, while adaptive retention is replaced by a fixed lifetime of 16 decoding steps. Removing candidate memory further eliminates these auxiliary KV slots and recovers standard CoT-Sampling.

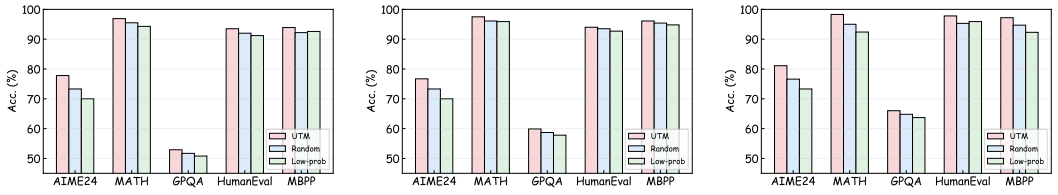
B Additional Experimental Results

B.1 Effect of Candidate KV Components

We examine whether the effectiveness of candidate memory depends on preserving complete candidate KV states. Specifically, we compare the default Complete KV configuration with K-only, V-only, and Random KV variants on Qwen3-8B. In K-only and V-only, the candidate values and

Table 5: Effect of candidate KV components on Qwen3-8B. Complete KV denotes the default UTM configuration that retains both the original candidate keys and values. For a fair comparison, Random KV uses norm-matched random states. Accuracy is reported in percentage, and AVG. denotes the macro-average across all benchmarks.

	Math Reasoning					Knowledge	Code Generation			Overall
Variant	**AIME24**	**AIME25**	**AMC23**	**GSM8K**	**MATH-500**	**GPQA**	**HumanEval**	**LCB**	**MBPP**	**AVG.**
Complete KV	76.7	75.6	95.8	95.9	97.5	59.9	94.0	60.8	96.1	83.6
K-only	66.7	53.3	94.2	91.2	89.6	56.4	92.7	56.7	94.3	77.2
V-only	63.3	63.3	92.5	92.7	92.0	57.1	93.3	55.8	94.8	78.3
Random KV	53.3	43.3	82.5	82.8	78.2	50.1	87.8	51.1	89.2	68.7



(a) Qwen3-4B (b) Qwen3-8B (c) Qwen3-32B

Figure 5: Effect of candidate selection strategies across model scales. Random candidates are sampled from ranks 5–20, while low-probability candidates are selected from ranks 20–50.

keys are zeroed, respectively, while all other UTM components remain unchanged. For a fair comparison, Random KV replaces the original candidate states with norm-matched random states.

As shown in Table 5, Complete KV consistently achieves the best performance across all nine benchmarks. Removing candidate values or keys decreases the average accuracy from 83.6% to 77.2% and 78.3%, respectively, showing that neither component alone is sufficient. Candidate keys enable query-dependent access to relevant memory states, whereas candidate values preserve the information retrieved during subsequent decoding. V-only performs slightly better on average than K-only, suggesting that candidate values retain useful content even without precise query–key matching. Nevertheless, its clear gap from Complete KV demonstrates the importance of selective retrieval through candidate keys. Random KV further reduces the average accuracy to 68.7%, showing that replacing candidate KV states with random states not only removes the meaningful information encoded in plausible alternatives, but also introduces distracting memory that may interfere with the information carried by the main reasoning path.

B.2 Effect of Candidate Selection

UTM constructs candidate memory from unchosen tokens in the decoding distribution. We investigate whether the choice of candidate states affects the effectiveness of memory augmentation. Specifically, we compare the top-ranked unchosen-token selection used in UTM with two alternative strategies. The random strategy samples candidates from ranks 5–20, while the low-probability strategy selects candidates from ranks 20–50 of the decoding distribution. For this analysis only, we increase the decoding top-k from 30 to 50 so that candidates up to rank 50 are available for selection; all other hyperparameters remain unchanged.

As shown in Figure 5, selecting top-ranked unchosen tokens consistently provides better performance than both alternatives across different model scales. Although random candidates are sampled from relatively plausible regions of the distribution, they are less effective than top-ranked unchosen tokens, while low-probability candidates provide limited benefit. This indicates that candidate states are not equally informative, and high-ranked candidates provide a better trade-off between candidate diversity and reliability.

Table 6: Effect of Top-k and Top-p candidate selection. Top-k retains the eight highest-probability unchosen tokens, whereas Top-p dynamically determines the candidate set using $p = 0.9$. Accuracy is reported in percentage, and AVG. denotes the macro-average across all benchmarks.

	Math Reasoning					Knowledge	Code Generation			Overall
Selection	**AIME24**	**AIME25**	**AMC23**	**GSM8K**	**MATH-500**	**GPQA**	**HumanEval**	**LCB**	**MBPP**	**AVG.**
					Qwen3-4B					
Top-k ($k = 8$)	77.8	71.1	98.3	95.9	96.9	52.9	93.5	56.2	93.9	81.8
Top-p ($p = 0.9$)	76.7	70.0	97.5	95.8	96.2	52.5	92.7	54.8	93.4	81.1
					Qwen3-8B					
Top-k ($k = 8$)	76.7	75.6	95.8	95.9	97.5	59.9	94.0	60.8	96.1	83.6
Top-p ($p = 0.9$)	76.7	73.3	90.0	95.9	97.4	61.6	93.3	57.7	95.7	82.4
					Qwen3-32B					
Top-k ($k = 8$)	81.1	80.0	98.3	96.9	98.3	66.0	97.8	65.0	97.2	86.7
Top-p ($p = 0.9$)	80.0	70.0	97.5	96.4	98.2	64.7	98.2	62.7	98.1	85.1

Table 7: Effect of candidate prior formulations on Qwen3-8B. Top-1 Relative denotes the default prior formulation used in UTM. Accuracy is reported in percentage, and AVG. denotes the macro-average across all benchmarks.

	Math Reasoning					Knowledge	Code Generation			Overall
Prior	**AIME24**	**AIME25**	**AMC23**	**GSM8K**	**MATH-500**	**GPQA**	**HumanEval**	**LCB**	**MBPP**	**AVG.**
Top-1 Relative (UTM)	76.7	75.6	95.8	95.9	97.5	59.9	94.0	60.8	96.1	83.6
Absolute Prior	66.7	56.7	87.5	92.7	94.2	57.6	88.7	57.2	93.7	77.2
No Prior	66.7	56.7	85.0	90.6	92.9	57.1	86.2	54.8	90.6	75.6

B.3 Effect of Top-k and Top-p Candidate Selection

We compare the fixed-count Top-k candidate selection used in UTM with a probability-mass-based Top-p variant. After excluding the token selected by the main decoding path, Top-k retains the eight highest-probability unchosen tokens, whereas Top-p selects a variable number of candidates according to a cumulative probability threshold of $p = 0.9$. The main-path decoding strategy and all other UTM components are kept unchanged to isolate the effect of candidate selection.

As shown in Table 6, Top-k achieves higher average accuracy than Top-p across all three model scales, with improvements from 81.1% to 81.8% on Qwen3-4B, from 82.4% to 83.6% on Qwen3-8B, and from 85.1% to 86.7% on Qwen3-32B. Although Top-p occasionally improves individual benchmarks, its performance is less consistent across tasks. These results suggest that retaining a fixed number of the highest-ranked alternatives provides a more stable candidate memory, whereas probability-mass-based selection may introduce varying numbers and qualities of candidate states across decoding steps.

B.4 Effect of Relative Prior Design

We examine the role of prior calibration by comparing the Top-1-relative prior used in UTM with an absolute-prior variant and a no-prior variant. The absolute prior directly adds the candidate log-probability to the memory attention logit, while the no-prior variant relies solely on query–key similarity. All other candidate construction, retention, and decoding settings are kept unchanged.

As shown in Table 7, the absolute prior generally improves over using no prior, indicating that the original candidate probability provides useful confidence information. However, directly using absolute probabilities remains sensitive to variations in probability scale across decoding steps. In contrast, the Top-1-relative prior calibrates each candidate against the highest-probability token at the same construction step, providing a more stable local reference. This design improves the average accuracy from 77.2% and 75.6% to 83.6%, and consistently achieves the strongest performance across all nine benchmarks.

Table 8: Efficiency comparison across nine benchmarks. Length denotes the average generated tokens per sample. Throughput is the total generated tokens divided by total wall-clock time for each benchmark, including run-level overhead. AVG. denotes the macro-average across benchmarks. All results are measured under the same configuration on four NVIDIA A800 GPUs.

Dataset	CoT-Sampling		Soft-Thinking		UTM	
	Length	Tokens/s	Length	Tokens/s	Length	Tokens/s
AIME24	14,845.6	406.7	14,748.9	370.5	15,510.2	347.5
AIME25	17,931.5	434.6	17,896.8	393.5	18,371.9	341.0
AMC23	8,154.4	392.6	9,015.8	336.0	7,823.8	326.8
GSM8K	1,831.0	448.9	1,688.3	412.4	1,854.0	319.4
MATH-500	6,418.6	191.9	5,066.0	168.9	6,311.0	170.9
GPQA	8,782.8	241.4	8,129.0	195.4	8,302.3	186.5
HumanEval	3,601.4	203.8	3,976.7	99.3	4,267.7	169.8
LCB	17,223.2	449.6	15,495.7	454.5	17,068.1	419.0
MBPP	3,197.5	215.0	3,145.8	233.8	2,945.8	229.3
AVG.	9,109.6	331.6	8,795.9	296.0	9,161.6	278.9

B.5 Efficiency Analysis

We further evaluate the efficiency of UTM in terms of generation length and end-to-end output throughput. All methods are measured under the same decoding configuration and hardware environment. For each benchmark, throughput is computed as the total number of generated tokens divided by the total wall-clock time of the run, which includes run-level initialization overhead. We report generation length together with throughput to distinguish the computational overhead introduced by UTM from differences in the number of generated tokens.

As shown in Table 8, UTM produces an average of 9,161.6 tokens per sample, which is comparable to 9,109.6 for CoT-Sampling and 8,795.9 for Soft-Thinking. Moreover, UTM does not consistently generate longer outputs across benchmarks, indicating that its accuracy improvements do not primarily result from extending the reasoning trajectory. UTM achieves an average throughput of 278.9 tokens per second, compared with 331.6 for CoT-Sampling and 296.0 for Soft-Thinking, corresponding to reductions of 15.9% and 5.8%, respectively. This overhead mainly arises from the additional forward computation required to construct candidate KV states and the enlarged attention context when candidate memory is accessed. Candidate storage and memory-augmented decoding limit these additional operations to selected decoding steps, while adaptive retention further bounds the size of active candidate memory.

C Case Study

Figure 6 presents a GSM8K example comparing the generated reasoning trajectories of standard CoT decoding and UTM. Standard CoT appends one selected token to the generated reasoning trajectory at each decoding step, while the remaining candidates are discarded. In this example, the generated trajectory interprets the late-checkout guests as still present at the final time point. Although the subsequent arithmetic is internally consistent, it subtracts only the early-checkout group and therefore produces the incorrect answer of 113.

UTM maintains a single generated reasoning trajectory while preserving the KV states of top-ranked unchosen tokens as auxiliary memory. These candidate states do not enter the visible reasoning trajectory or generate additional continuations, but remain accessible to subsequent decoding steps through attention. In this case, the UTM response accounts for both the early- and late-checkout groups in the final count and obtains the correct answer of 98. This example is consistent with our claim that information associated with unchosen tokens can remain useful beyond the decoding steps at which they are produced.

Case Study: Candidate Memory in Mathematical Reasoning

Problem: Hotel Checkout

A hotel has 100 guests in the morning. Among them, 24 choose an early checkout and 15 choose a late checkout. In the afternoon, twice as many guests as the late-checkout group check in, and another 7 guests arrive after dinner. How many guests are staying at the hotel now?

CoT (Incorrect) ✗

Reasoning Path

There are initially 100 guests.

After 24 guests complete an early checkout,

$$100 - 24 = 76.$$

The 15 late-checkout guests are interpreted as still staying at the hotel, so they are not subtracted.

The number of afternoon check-ins is

$$2 \times 15 = 30.$$

After another 7 guests arrive,

$$76 + 30 + 7 = 113.$$

Answer: 113 ✗

Analysis. CoT fails to subtract the late-checkout group, leading to the incorrect answer 113.

UTM (Ours) ✓

Unchosen Token Memory

At the key ambiguous step, UTM retains KV states associated with plausible unchosen alternatives:

$$\underbrace{\text{“still staying”}}_{\substack{\text{generated}\\ \text{trajectory}}} \qquad \underbrace{\text{“departed”}}_{\substack{\text{candidate}\\ \text{memory}}}$$

The retained states remain accessible to subsequent decoding steps through attention.

Reasoning Path

There are initially 100 guests.

By the final time point, both the early- and late-checkout groups have departed. Hence, the total number of checkouts is

$$24 + 15 = 39.$$

The number of remaining guests is

$$100 - 39 = 61.$$

The number of afternoon check-ins is

$$2 \times 15 = 30.$$

After another 7 guests arrive,

$$61 + 30 + 7 = 98.$$

Answer: 98 ✓

Analysis. UTM accounts for both checkout groups and produces the correct answer 98.

Retained unchosen-token states can provide useful information for subsequent reasoning.

Figure 6: A GSM8K case comparing the reasoning trajectories of standard CoT decoding and UTM.