THINKING IN LATENTS: ADAPTIVE ANCHOR REFINEMENT FOR IMPLICIT REASONING IN LLMS

Disha Sheshanarayana[1] **Rajat Subhra Pal**[2] **Manjira Sinha**[2] **Tirthankar Dasgupta**[2]

[1]Manipal University Jaipur
[2]TCS Research

disha.229301161@muj.manipal.edu,
{rajat.pal, sinha.manjira, dasgupta.tirthankar}@tcs.com

ABSTRACT

Token-level Chain-of-Thought (CoT) prompting has become a standard way to elicit multi-step reasoning in large language models (LLMs), especially for mathematical word problems. However, generating long intermediate traces increases output length and inference cost, and can be inefficient when the model could arrive at the correct answer without extensive verbalization. This has motivated latent-space reasoning approaches that shift computation into hidden representations and only emit a final answer. Yet, many latent reasoning methods depend on a fixed number of latent refinement steps at inference, adding another hyperparameter that must be tuned across models and datasets to balance accuracy and efficiency. We introduce AdaAnchor, a latent reasoning framework that performs silent iterative computation by refining a set of latent anchor vectors attached to the input. AdaAnchor further incorporates an adaptive halting mechanism that monitors anchor stability across iterations and terminates refinement once the anchor dynamics converge, allocating fewer steps to easier instances while reserving additional refinement steps for harder ones under a shared maximum-step budget. Our empirical evaluation across three mathematical word-problem benchmarks shows that AdaAnchor with adaptive halting yields accuracy gains of up to 5% over fixed-step latent refinement while reducing average latent refinement steps by 48–60% under the same maximum-step budget. Compared to standard reasoning baselines, AdaAnchor achieves large reductions in generated tokens (92–93%) by moving computation into silent latent refinement, offering a different accuracy–efficiency trade-off with substantially lower output-token usage.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated strong capabilities in mathematical reasoning, particularly when prompted to produce explicit intermediate traces such as Chain-of-Thought (CoT) (Wei et al., 2022). Recent progress has further amplified these gains through improved instruction-tuning and longer reasoning trajectories, reinforcing the view that additional thinking tokens can unlock better problem-solving behavior (Kojima et al., 2022; Wang et al., 2022). Despite these advances, a practical limitation persists: lengthy reasoning traces are computationally expensive, increasing decoding latency and token usage and raising serving cost especially under high-concurrency deployments. This trade-off motivates methods that preserve the benefits of multi-step reasoning while reducing the cost of token-level generation.

Several research studies have emerged to address this computational challenge. One line of research focuses on improving efficiency within the token level, for example by encouraging more concise rationales, skipping less informative tokens, or terminating early when the model appears confident in a candidate answer (Goyal et al., 2024). While valuable, these methods remain coupled to sparse, discrete token generation and therefore inherit the cost structure of autoregressive decoding. A more promising direction explores reasoning within the dense latent space, where models can perform computation internally and emit only the final answer. Prior work has pursued latent reasoning via distillation and curriculum strategies, by reusing computation through layer skipping or looping

(Saunshi et al., 2025), or by partially replacing token-level traces with latent representations (Hao et al., 2024; Zelikman et al., 2024; Deng et al., 2023; Shen et al., 2025). However, many latent reasoning methods still depend on a fixed number of latent refinement steps at inference, introducing another hyperparameter that must be tuned across models and datasets to balance accuracy and efficiency (Hao et al., 2024; Deng et al., 2023; 2024; Shen et al., 2025).

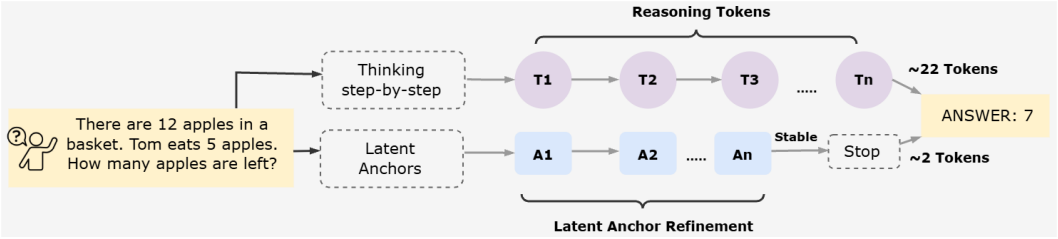

Figure 1: Comparison of AdaAnchor with explicit Chain-of-Thought (CoT) reasoning. CoT generates long intermediate reasoning tokens, whereas AdaAnchor performs implicit multi-step computation by refining latent anchor vectors and uses stability-based early stopping before answer-only decoding.

To overcome these limitations, we introduce AdaAnchor, an implicit reasoning framework that performs silent iterative computation by refining a compact set of latent anchor vectors attached to the input. Instead of generating intermediate reasoning tokens, AdaAnchor updates these anchors using the model's hidden activations while keeping the output in an answer-only format. Furthermore, we propose an adaptive halting strategy for latent refinement: AdaAnchor tracks anchor stability across iterations and stops updating when the dynamics converge, thereby allocating computation on a per-instance basis. This enables instance-wise compute allocation under a shared maximum-step budget, so easier problems terminate quickly while harder problems receive additional refinement without requiring a fixed latent step count to be tuned per dataset (Graves, 2016).

We evaluate AdaAnchor on three mathematical word-problem benchmarks, GSM8K, SVAMP, and MultiArith, and compare it against fixed-step latent refinement and standard reasoning baselines under a shared maximum latent budget (Cobbe et al., 2021; Patel et al., 2021; Roy & Roth, 2015). The results show that convergence-driven adaptive halting improves the efficiency–accuracy trade-off relative to fixed-step refinement by avoiding unnecessary latent iterations, while answer-only decoding reduces generation overhead compared to token-based baselines. Together, these findings indicate that stability-aware termination is a practical mechanism for controlling implicit computation and improving the deployability of latent reasoning methods.

2 Related Work

2.1 Explicit CoT reasoning

Large Language Models (LLMs) often exhibit substantially stronger mathematical reasoning when prompted to generate explicit intermediate traces, most notably through Chain-of-Thought (CoT) style prompting Wei et al. (2022). Follow-up work showed that even without exemplars, carefully designed prompts can elicit multi-step reasoning behaviors Kojima et al. (2022), and decoding-time strategies such as self-consistency can further improve reliability by aggregating over diverse reasoning paths Wang et al. (2022). At the same time, generating long token-level rationales can be costly in practice, motivating methods that reduce the number of produced tokens while preserving reasoning quality. Representative directions include introducing structured pause mechanisms that encourage internal computation without fully verbalizing every step Goyal et al. (2024), as well as prompting frameworks that constrain or compress intermediate traces to be shorter and cheaper to decode Aytes et al. (2025); Xu et al. (2025). Orthogonally, improvements at the training objective level—such as predicting multiple future tokens per position have been explored as a way to increase throughput and reduce inference time without changing the basic autoregressive interface Gloeckle et al. (2024).

2.2 LATENT REASONING

To avoid the overhead of long textual traces, latent reasoning approaches shift more computation into hidden representations and emit only the final answer. A prominent line of work performs silent reasoning by feeding internal hidden states back into the model as continuous thoughts, enabling multi-step computation without committing to discrete tokens at every step Hao et al. (2024). Related approaches learn latent reasoning trajectories via distillation or self-training so that intermediate computation is represented implicitly rather than as explicit text Shen et al. (2025). More recently, hybrid schemes have explored mixing latent and text tokens during training and inference, aiming to retain some controllability and interpretability while still gaining efficiency from latent computation Su et al. (2025). In parallel, latent-space compression methods seek to reduce the length of reasoning traces by compressing multiple reasoning tokens into fewer latent steps, improving efficiency while keeping performance competitive Tan et al. (2025). Another complementary direction analyzes or exploits latent dynamics during generation to guide or improve reasoning behavior Zelikman et al. (2024).

2.3 ADAPTIVE HALTING AND COMPUTE ALLOCATION

A recurring limitation in iterative (token or latent) reasoning is that many methods require fixing the number of refinement steps in advance, which can lead to over-computation on easy instances and under-computation on hard ones. Adaptive computation mechanisms address this by allowing models to halt dynamically based on instance difficulty, with classic formulations such as Adaptive Computation Time (ACT) learning when to stop iterating Graves (2016).

Prior work on latent and implicit reasoning often (i) runs a fixed number of silent refinement steps, (ii) relies on task-specific training signals to control computation, or (iii) lacks explicit budget control. In contrast, AdaAnchor introduces a compact latent bottleneck via learnable anchor vectors and a convergence-based halting rule, enabling per-example adaptive compute under a shared maximum-step budget without training a separate halting controller.

3 METHOD

3.1 PROBLEM FORMULATION

We consider mathematical word-problem solving in an answer-only setting. Given an input question x (a sequence of tokens), the goal is to predict the correct final answer y. Standard reasoning prompting often generates an intermediate rationale r before producing the answer. While such explicit traces can improve accuracy, they incur substantial inference cost because both the rationale and the answer must be generated autoregressively, increasing latency and token usage.

Our objective is to retain the benefits of multi-step reasoning while reducing the overhead of token-level generation. We work with a base autoregressive transformer f_θ and introduce a compact latent state in the form of anchor vectors. Let $A^{(0)} \in \mathbb{R}^{m \times d}$ denote m anchor embeddings of dimension d. We augment the model input in embedding space by prepending projected anchor embeddings to the token embeddings, denoted $[P(A^{(t)}); Emb(x)]$. Instead of decoding a long rationale, the model performs silent iterative computation by repeatedly updating the anchor state for a variable number of refinement steps T, producing a sequence $\{A^{(t)}\}_{t=0}^{T}$. After refinement terminates, the model generates the final answer $\hat{y}$ conditioned on the refined anchors and the original question, while keeping the output in an answer-only format.

We measure efficiency along two axes that directly affect inference cost: (i) the number of refinement iterations used per instance, and (ii) the number of generated output tokens. The core problem is therefore to design (a) an anchor-based refinement mechanism that supports implicit multi-step computation and (b) an instance-wise stopping rule that selects $T \leq K_{\max}$ automatically, so that the model avoids unnecessary refinement on easy inputs while allocating additional latent computation to harder ones. The AdaAnchor refinement procedure and the stability-based halting rule are described in Sections 3.2 and 3.3, respectively.

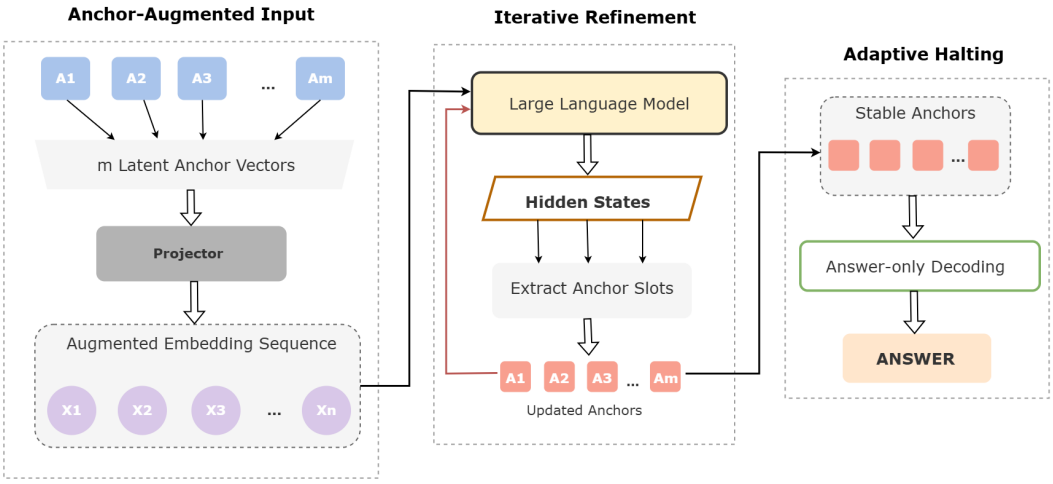

Figure 2: **Overview of AdaAnchor.** AdaAnchor prepends m learnable latent anchor vectors to the input embedding sequence (left), iteratively refines them via repeated forward passes and anchor-slot updates (middle), and uses a stability-based criterion to halt early before performing answer-only decoding (right).

3.2 AdaAnchor Framework

We propose AdaAnchor, a latent reasoning framework that enables implicit multi-step computation in large language models without generating explicit intermediate reasoning tokens (Figure 2). AdaAnchor augments a base autoregressive transformer f_θ with a compact set of learnable anchor vectors prepended to the input in embedding space. These anchors act as a reusable low-dimensional state that is iteratively refined through repeated forward passes, allowing the model to think silently while keeping the textual interface in an answer-only format.

Anchor-augmented input. Let the tokenized question be $x = (x_1, \ldots, x_n)$, and let $\mathrm{Emb}(x) \in \mathbb{R}^{n \times d}$ denote its input embedding sequence (hidden size d). AdaAnchor maintains m anchor vectors

$$A^{(t)} = [a_1^{(t)}, \ldots, a_m^{(t)}] \in \mathbb{R}^{m \times d}, \tag{1}$$

where t indexes the refinement iteration. At inference, we initialize anchors with the learned parameters, i.e., $A^{(0)} \leftarrow A_{\text{learned}}$. At iteration t, we form the augmented embedding sequence

$$E^{(t)} = [\tilde{A}^{(t)};\, \mathrm{Emb}(x)], \qquad \tilde{A}^{(t)} = P(A^{(t)}), \tag{2}$$

where $P(\cdot)$ projects anchor vectors into the same embedding space as token embeddings.

Prior soft-prompting or prefix-tuning methods learn a set of static prompt embeddings that are prepended to the input and remain fixed throughout inference (Lester et al., 2021; Li & Liang, 2021). AdaAnchor's anchors are different: they form an explicit latent state $A^{(t)}$ that is iteratively rewritten during inference. After each forward pass on $[P(A^{(t)}); \mathrm{Emb}(x)]$, we update the anchor slots using the model's hidden states and feed the refined anchors into the next refinement step. In this way, anchors act as a persistent latent memory across refinement iterations rather than a fixed learned prefix, enabling silent multi-step computation under a shared refinement budget.

Iterative anchor refinement. Given $E^{(t)}$, we run a forward pass through the base LM to obtain

$$H^{(t)} = f_\theta(E^{(t)}) \in \mathbb{R}^{(m+n) \times d}, \tag{3}$$

where $H^{(t)}$ denotes the final-layer hidden states for all positions in the augmented sequence. AdaAnchor updates the anchors by extracting the hidden states corresponding to the anchor positions.

When $\beta = 1$, the update reduces to overwriting; smaller β smooths anchor evolution and improves convergence behavior in practice. The refinement loop runs for at most $K_{\max}$ iterations, and it can terminate early via the adaptive halting criterion described in Section 3.3.

Algorithm 1 AdaAnchor Inference with Adaptive Halting

Require: Tokenized question x, learned anchor parameters $A_{\text{learned}} \in \mathbb{R}^{m \times d}$, max refinement budget $K_{\max}$, smoothing $\beta \in (0, 1]$, threshold τ, patience s

Set anchors $A^{(0)} \leftarrow A_{\text{learned}}$
$c \leftarrow 0$ {consecutive stable counter}
for $t = 0, 1, \ldots, K_{\max} - 1$ **do**
  Form augmented input $E^{(t)} \leftarrow [\, P(A^{(t)});\ \mathrm{Emb}(x)\,]$
  Run backbone LM on $E^{(t)}$ to obtain hidden states $H^{(t)}$
  Extract anchor-position states $A_{\text{new}}^{(t+1)} \leftarrow H_{1:m}^{(t)}$
  Smooth update $A^{(t+1)} \leftarrow (1 - \beta)A^{(t)} + \beta A_{\text{new}}^{(t+1)}$
  Compute stability $\Delta^{(t+1)} \leftarrow 1 - \cos(\bar{a}^{(t+1)}, \bar{a}^{(t)})$, where $\bar{a}^{(t)} = \frac{1}{m}\sum_{i=1}^{m} a_i^{(t)}$ and $a_i^{(t)}$ denotes the i-th anchor vector in $A^{(t)}$
  if $\Delta^{(t+1)} < \tau$ **then**
    $c \leftarrow c + 1$
  else
    $c \leftarrow 0$
  end if
  if $c \geq s$ **then**
    break
  end if
end for
Decode answer-only $\hat{y}$ conditioned on $[\, P(A^{(t+1)});\ \mathrm{Emb}(x)\,]$
return $\hat{y}$

Answer-only decoding. After refinement terminates at step T, AdaAnchor generates the final answer by decoding only a short answer continuation conditioned on the refined anchors and the original input. By performing computation through latent refinement rather than token-level rationales, AdaAnchor substantially reduces generated tokens while keeping the output concise.

3.3 Adaptive Halting

We incorporate an adaptive halting mechanism that determines how many latent refinement steps to run per instance. Since AdaAnchor performs iterative computation by updating the anchor state $A^{(t)}$, we use anchor stability across iterations as a convergence signal. Intuitively, when refinement is still productive, consecutive anchor states change noticeably; once refinement has converged, anchor updates become small and repetitive.

Anchor stability metric. We quantify refinement progress by measuring the change between successive anchor states. In particular, we summarize the anchors at iteration t by their mean representation

$$\bar{a}^{(t)} = \frac{1}{m}\sum_{i=1}^{m} a_i^{(t)}, \tag{4}$$

and define a scalar update magnitude via cosine distance dissimilarity:

$$\Delta^{(t)} = 1 - \cos\big(\bar{a}^{(t)}, \bar{a}^{(t-1)}\big). \tag{5}$$

Smaller $\Delta^{(t)}$ indicates that the anchor dynamics are approaching a fixed point, suggesting that further refinement is unlikely to add useful computation. (An equivalent alternative is to average the change across all anchors; we adopt this compact summary for robustness and efficiency.)

Halting rule. AdaAnchor refines anchors for at most $K_{\max}$ iterations, but it can stop early when the anchor updates converge. Concretely, we halt at the first iteration T such that the update magnitude remains below a threshold τ for s consecutive steps:

$$T = \min\Big\{ t \in \{1, \ldots, K_{\max}\} \ :\ \Delta^{(t-j)} < \tau \ \ \forall j \in \{0, \ldots, s-1\} \Big\}. \tag{6}$$

This yields instance-wise compute allocation under a shared maximum-step budget: easier instances typically converge in fewer steps, while harder ones continue refining until convergence or the budget limit. The halting check introduces negligible overhead because it operates directly on anchor states already computed during refinement.

4 EXPERIMENTS

In this section, we evaluate our method AdaAnchor on mathematical word-problem solving benchmarks and compare it against fixed-step latent refinement and standard reasoning baselines under a shared maximum latent budget.

4.1 EXPERIMENTAL SETUP

4.1.1 DATASETS

Our training is conducted primarily on **GSM8K** and **SVAMP**. Specifically, we use approximately 7.47K training examples from GSM8K and 700 training examples from SVAMP to fine-tune the models and learn the AdaAnchor components. We evaluate the trained models on three mathematical word-problem benchmarks: **GSM8K**, **SVAMP**, and **MultiArith**. The corresponding test sets contain 1.32K, 300, and 600 examples, respectively. For all datasets, we use an answer-only setup, where the model is prompted with the question and is expected to output only the final numeric answer.

Table 1: Overview of datasets used.

Dataset	Task Type	Metric
GSM8K(Cobbe et al., 2021)	Grade-school mathematical word problems	Accuracy
SVAMP(Patel et al., 2021)	Arithmetic word problems	Accuracy
MultiArith(Roy & Roth, 2015)	Multi-step arithmetic word problems	Accuracy

4.1.2 MODELS

We train and evaluate on two backbone language models, **Qwen2.5-1.5B**(Qwen Team et al., 2024) and **Llama-3.2-1B**(Grattafiori et al., 2024) parameters, across all experiments. We focus on small LMs, where efficiency differences from latent refinement are easier to observe and measure reliably. Throughout our experiments, we use deterministic decoding and hold inference settings constant across methods to ensure a fair comparison.

4.1.3 BASELINES AND METRICS

Baselines. We compare AdaAnchor against token-based and latent reasoning baselines that reflect common evaluation settings for efficient reasoning. (1) **No CoT**: the model is prompted to directly output the final answer in an answer-only format, without generating intermediate reasoning. (2) **CoT**: the model generates an explicit step-by-step rationale followed by the final answer using standard Chain-of-Thought prompting. (3) **iCoT**: an implicit-CoT style baseline that removes explicit reasoning traces while retaining an answer-only output format, serving as a lightweight implicit reasoning comparison.

Metrics. We evaluate using three metrics: **Accuracy** (exact-match correctness of the final answer under deterministic decoding), **Average Tokens** (average number of generated output tokens per example), and **Average Steps** (average number of latent refinement iterations executed per example).

4.1.4 IMPLEMENTATION DETAILS

During training, we keep the backbone LM weights frozen and optimize only the AdaAnchor-specific components (learnable anchor embeddings and the small projection used to inject anchors), together with low-rank adaptation modules (LoRA) (Hu et al., 2021). We train for 20 epochs using

		GSM8K			SVAMP			MultiArith		
Model	**Method**	**Acc.**	**Avg Tok**	**Avg Steps**	**Acc.**	**Avg Tok**	**Avg Steps**	**Acc.**	**Avg Tok**	**Avg Steps**
Qwen2.5-1.5B	No CoT	13.0	2.16	–	42.0	2.34	–	22.3	2.41	–
	CoT	20.0	28.27	–	59.3	29.09	–	34.3	30.2	–
	iCoT	12.23	2.36	–	48.5	2.04	–	28.56	1.66	–
	AdaAnchor (K=8)	16.0	2.73	8	50.5	2.12	8	27.6	2.34	8
	AdaAnchor (adaptive)	16.0	2.17	3.23	55.2	2.23	4.12	29.4	2.16	3.82
Llama-3.2-1B	No CoT	10.5	2.98	–	38.2	2.10	–	20.56	2.08	–
	CoT	23.2	25.4	–	57.8	28.21	–	43.33	28.0	–
	iCoT	11.7	2.25	–	54.2	2.43	–	30.84	2.12	–
	AdaAnchor (K=8)	14.0	2.89	8	52.0	2.13	8	28.31	2.48	8
	AdaAnchor (adaptive)	17.2	2.45	3.5	53.4	2.8	3.1	32.44	2.57	3.5

Table 2: Experimental results on mathematical reasoning benchmarks. We report accuracy (Acc. %), average output tokens (Avg Tok), and average latent refinement steps (Avg Steps) for answer-only evaluation. No CoT denotes direct answer generation without reasoning. CoT generates full chain-of-thought rationales before the answer. iCoT uses implicit chain-of-thought.

mixed precision. Optimization uses AdamW (Loshchilov & Hutter, 2017) with a fixed learning rate of $1e{-}4$ and weight decay $1e{-}2$, per-device batch size 1, and gradient accumulation of 16; we select the final checkpoint based on validation accuracy. In addition to the answer-only loss, we include an auxiliary anchor-alignment objective derived from coarse chunks of the rationale text, with dataset-dependent weighting as implemented in our code. At inference, AdaAnchor refines anchors up to a maximum budget using a smoothed update, and applies stability-based early stopping by monitoring anchor change across iterations and halting once it consistently converges.

4.2 Results

Table 2 compares AdaAnchor against standard baselines on three mathematical word-problem benchmarks, GSM8K, SVAMP, and MultiArith, under an answer-only evaluation setup. Across datasets, AdaAnchor consistently improves over No-CoT prompting, delivering relative accuracy gains of roughly $\sim$23–32% on Qwen2.5-1.5B and $\sim$39–64% on Llama-3.2-1B, while keeping output-token usage very low. In particular, compared to explicit CoT, AdaAnchor reduces generated tokens by about $\sim$90–93%, highlighting a different accuracy–efficiency trade-off in which more computation is shifted into silent latent refinement rather than verbose token-level rationales.

Moreover, the adaptive halting variant further improves this trade-off by avoiding unnecessary refinement once anchor dynamics stabilize. Relative to a fixed refinement budget ($K = 8$), adaptive stopping uses $\sim$48–61% fewer latent refinement steps on average while maintaining similar accuracy and improving it in several cases, indicating effective instance-wise compute allocation: easier instances terminate early, while harder ones continue refining under the same maximum-step budget. Overall, these results suggest that convergence-aware latent refinement can support iterative reasoning with substantially lower output-token usage and fewer refinement steps.

4.3 Ablation Study

We conduct an ablation study to isolate the contribution of stability-based adaptive halting in AdaAnchor. Specifically, we compare a fixed-step refinement variant that always runs a constant number of latent refinement iterations ($K = 8$) against the adaptive variant that refines anchors up to the same maximum budget ($K_{\max} = 8$) but halts early once anchor updates converge. Both variants use the same anchor design, refinement update rule, and decoding configuration; the only difference is whether refinement proceeds for a fixed number of iterations or terminates based on stability.

To understand how performance depends on the available refinement budget, we vary the fixed-step refinement length on the Qwen2.5-1.5B using $K \in \{1, 2, 4, 8\}$ as shown in Figure 3. Accuracy generally improves as the budget increases, but the gains saturate beyond moderate values, indicating diminishing returns from always executing a large fixed K. This supports the motivation for convergence-aware termination, which can avoid unnecessary refinement when additional iterations provide limited benefit.

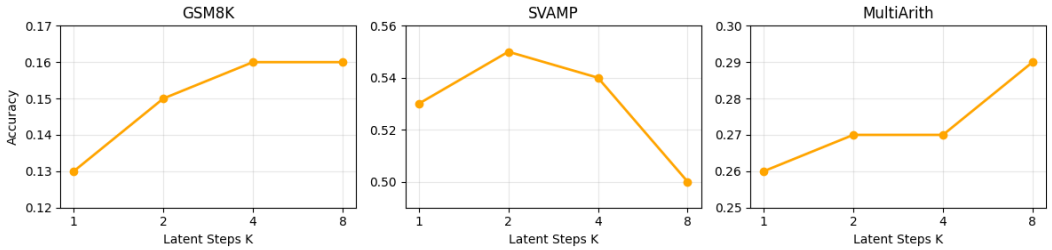

Figure 3: Accuracy vs. fixed latent refinement budget K on Qwen2.5-1.5B. Each panel reports performance as a function of $K \in \{1, 2, 4, 8\}$.

We also analyze the refinement lengths selected by adaptive halting on Qwen2.5-1.5B under the shared maximum budget $K_{\max} = 8$ as shown in Figure 4. The halting-step distribution shows that the model frequently stops well before reaching the maximum budget, while reserving more refinement steps for a smaller fraction of harder instances. This behavior confirms that the stability criterion induces instance-wise compute allocation in practice and explains the reduction in average latent steps relative to fixed-step refinement without tuning a fixed latent-step hyperparameter per dataset.

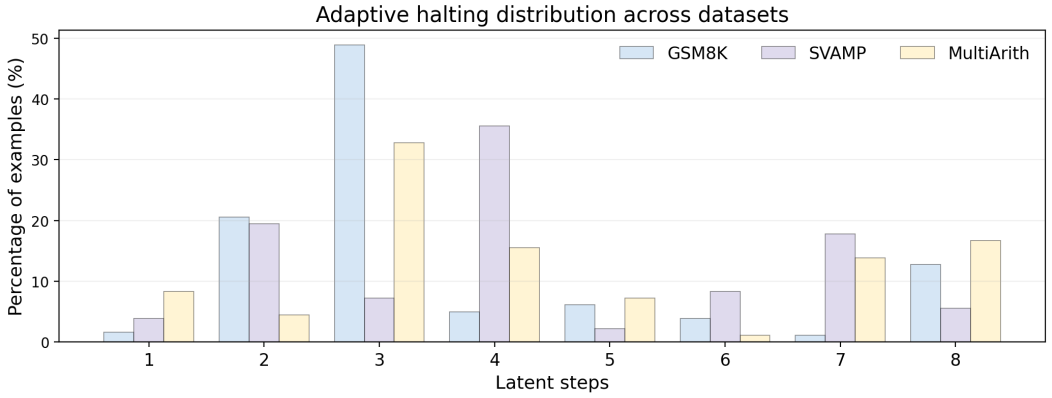

Figure 4: Adaptive halting distribution showing percentage of examples halting at each step (1-8) across datasets on Qwen2.5-1.5B.

5 Limitations and Future Work

While AdaAnchor improves the efficiency–accuracy trade-off via stability-based adaptive halting, it has few limitations. First, our halting mechanism relies on a hand-designed stability criterion. Although this heuristic performs well in our setting, it may be sensitive to hyperparameter choices and can occasionally halt too early or too late on atypical inputs or under distribution shifts. Second, while anchor refinement provides a compact latent reasoning state, the semantics of the learned anchors are not directly interpretable, and it remains difficult to precisely attribute improvements to specific latent behaviors compared to explicit token-level rationales.

These limitations motivate several directions for future work. A natural extension is to replace the heuristic stopping rule with a learned halting policy, for example via a lightweight controller trained with supervision or reinforcement learning, or by incorporating calibrated confidence/verification signals to make termination more robust across models and datasets. In addition, improving interpretability of anchor dynamics is an important direction: future work could introduce probing and visualization tools for anchor trajectories, enforce structured anchors aligned to intermediate

quantities , or add auxiliary objectives that encourage anchors to correspond to human-interpretable sub-computations.

6 CONCLUSION

We introduced AdaAnchor, a latent reasoning framework that performs multi-step computation by iteratively refining a compact set of learnable anchor vectors while keeping the model output in an answer-only format. By shifting reasoning from token-level traces into a small latent state, AdaAnchor targets lower output-token usage without requiring long intermediate generations. A key component is stability-based adaptive halting, which monitors anchor dynamics and terminates refinement once updates converge, enabling instance-wise allocation of latent computation under a shared maximum budget.

Empirically, AdaAnchor improves the efficiency–accuracy trade-off relative to fixed-step latent refinement while also substantially reducing output-token usage compared to token-level reasoning baselines. Under the same maximum latent budget, adaptive halting improves accuracy by up to $\sim$5% over fixed-step refinement while reducing the average number of latent refinement iterations by $\sim$48–60%. In addition, by emitting answer-only outputs and performing computation silently in latent space, AdaAnchor reduces generated tokens by $\sim$92–93% compared to token-level reasoning baselines, illustrating how latent refinement can support iterative reasoning with fewer generated tokens and fewer refinement steps.

REFERENCES

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 24296–24320, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1236. URL https://aclanthology.org/2025.emnlp-main.1236/.

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. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024. doi: 10.48550/arXiv.2404.19737.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.02226.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783. URL https://arxiv.org/abs/2407.21783.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

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.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. doi: 10.48550/arXiv.2106.09685. URL https://arxiv.org/abs/2106.09685.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. doi: 10.48550/arXiv.1711.05101. URL https://arxiv.org/abs/1711.05101.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. doi: 10.48550/arXiv.2412.15115. URL https://arxiv.org/abs/2412.15115.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1743–1752, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1202.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. Token assorted: Mixing latent and text tokens for improved language model reasoning. *arXiv preprint arXiv:2502.03275*, 2025. doi: 10.48550/arXiv.2502.03275.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. Think silently, think fast: Dynamic latent compression of llm reasoning chains. *arXiv preprint arXiv:2505.16552*, 2025. doi: 10.48550/arXiv.2505.16552.

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

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.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms. *arXiv preprint arXiv:2502.12134*, 2025. doi: 10.48550/arXiv.2502.12134.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.