CIRF: Tokenizing Chain-of-Thoughts into Reusable Functional Units for Efficient Latent Reasoning in Large Language Models

Anonymous ACL submission

Abstract

Implicit Chain-of-Thought (CoT) reduces the inference cost of large language models by internalizing the explicit rationales. However, existing approaches typically lack alignment with explicit rationales and adaptivity to example complexity. In this work, we propose CIRF (*Chain-of-thoughts Into Reusable Functional units*), an implicit CoT framework that performs reasoning as a dynamic sequence of discrete functional tokens. CIRF assigns a functional token to each semantically coherent reasoning unit in explicit CoT traces. The model is then fine-tuned to autoregressively generate functional tokens and their optional results, followed by the final answer. This design aligns latent reasoning with a sequence of functional units, facilitating parallel training, explicit rationale alignment, and adaptive reasoning. Extensive experiments on mathematical, symbolic, and commonsense reasoning benchmarks show that CIRF provides a favorable accuracy-latency trade-off compared with state-of-the-art implicit CoT methods. Further analyses demonstrate that CIRF constructs distinct, interpretable functional tokens, leading to consistent performance improvements.

1 Introduction

Chain-of-thought (CoT) reasoning (Wei et al., 2022; Wang et al., 2023) has become a widely used strategy for eliciting multi-step reasoning in large language models (LLMs). By prompting or training a model to generate intermediate rationales before producing the final answer, CoT has substantially improved performance on mathematical and symbolic reasoning tasks. Recent analysis (Sprague et al., 2025) suggests that the gains from CoT are strongest in domains that require symbolic or multi-step computation, further highlighting the importance of traceable intermediate steps for solving complex problems.

Despite its effectiveness, the latency and memory costs incurred by long reasoning traces have motivated research on implicit CoT (Deng et al., 2024; Hao et al., 2025). Instead of generating full rationales in natural language, these studies aim to internalize, compress, or replace explicit reasoning traces using hidden states or special tokens. They have shown that CoT can be performed implicitly within models, thereby reducing the cost of generating additional natural language tokens.

A key requirement for successful internalization is to align the LLM states during implicit CoT with those induced by explicit CoT. Typical implicit CoT methods utilize repeated placeholder tokens (Goyal et al., 2024; Xu et al., 2025b) for alignment. However, these methods require the model to organize long, dynamic reasoning traces over a sequence whose surface form is largely homogeneous, whereas typical autoregressive LLMs are trained to reason conditioned on growing, changing textual prefixes. Whether such identical carriers provide an effective substrate for multi-step reasoning remains underexplored.

To address this issue, several methods (Hao et al., 2025; Shen et al., 2025) autoregressively feed output hidden representations as the next token embeddings, thereby reflecting changes in input token states as reasoning proceeds. However, training with recursively fed hidden representations incurs substantial training cost and instability, particularly in modeling multiple implicit tokens. In their formulations, input representations at each step need to be sequentially generated in training, because they are unknown until the model finishes computation at the previous step. Furthermore, their training procedures typically follow long curriculum stages or require a teacher model. The difficulty in scaling the number of implicit tokens may limit performance, considering that recent work (Hassid et al., 2025) increasingly shows that the number of reasoning tokens is important: too few tokens can

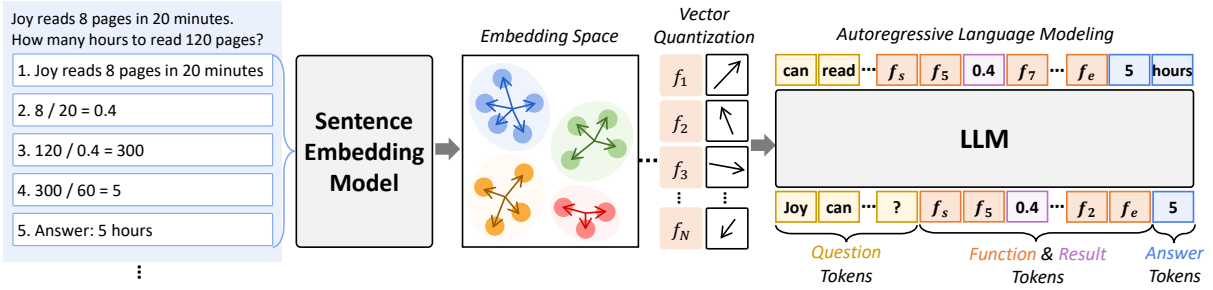

Figure 1: Overview of CIRF. Each functional unit in a CoT rationale is encoded using a sentence embedding model, mean-centered to reduce question-specific situational bias, and quantized into a discrete functional token. The target LLM is then fine-tuned to generate a compact sequence consisting of functional tokens, optional result tokens, and the final answer tokens. CIRF enables efficient latent reasoning through discrete functional tokens.

lead to underthinking, while too many can cause overthinking, instability, or wasted computation.

We propose a new implicit CoT framework, called CIRF (*Chain-of-thoughts Into Reusable Functional units*), which represents reasoning as an autoregressively generated sequence of discrete functional tokens. The key idea is to tokenize explicit CoT traces into functional units. Consequently, each rationale is converted into a short, dynamic sequence of function-bearing tokens, enabling parallel training over sequences whose lengths adapt to the complexity of the reasoning process. We then fine-tune a single universal LLM to generate these implicit functional tokens, followed by optional results and the final answer for alignment, across diverse reasoning datasets. This design addresses the above issues by utilizing semantically grounded and adaptively allocated functional tokens.

Through extensive experiments, we empirically verify that CIRF is more efficient than state-of-the-art implicit CoT methods across diverse reasoning benchmarks. We evaluate CIRF in terms of accuracy-latency trade-off with diverse configurations partially mixing implicit and explicit tokens. The results show that CIRF achieves the accuracy–latency frontier. In-depth analyses further reveal that CIRF provides interpretable functional token assignment and adaptive token sequences, leading to substantial performance improvements.

2 Related Work

2.1 Explicit CoT

It has been observed that LLMs can generate intermediate reasoning steps that improve the accuracy of final answers (Wei et al., 2022). Zero-shot CoT (Kojima et al., 2022) has shown that simple verbal triggers can elicit step-by-step reasoning without manually designed exemplars. While these findings highlight the importance of traceable intermediate steps for solving complex problems, many studies (Hassid et al., 2025; Wu et al., 2026) have also reported an inverted-U relationship between CoT length and accuracy: reasoning traces that are either too short or too long can hurt performance. Several studies (Xu et al., 2025a; Xia et al., 2025) have proposed methods for identifying shorter rationales that can match or exceed verbose CoT sequences. These studies are related to CIRF in that they show both the efficacy of generating intermediate reasoning steps and the importance of reasoning length for accuracy and efficiency.

2.2 Implicit CoT

Implicit CoT approaches aim to internalize reasoning into model hidden states. Early approaches have trained the hidden states associated with answer tokens to mimic the hidden states of explicit CoT (Deng et al., 2024). Placeholder tokens (Goyal et al., 2024; Pfau et al., 2024) have also been used as learnable thinking symbols. However, these methods may limit the reasoning capabilities of LLMs due to the lack of semantically grounded tokens. Another line of work has conducted reasoning through recursively fed output hidden states (Hao et al., 2025; Shen et al., 2025). However, these approaches have introduced substantial complexity in training and offered limited adaptivity in reasoning length. A further subgroup of research has used auxiliary models to generate compact reasoning surrogates (Xu et al., 2025b; He et al., 2025; Wei et al., 2025), causing additional cost from the auxiliary models. Compared with these methods,

CIRF provides a more efficient, straightforward strategy via a sequence of discrete functional tokens: facilitating single stage training with teacher forcing, realizing fine-grained reasoning functions and inducing adaptive reasoning behavior in a single unified model.

2.3 Tokenization of Latent Representations

Wang et al. (2024) have explicitly inferred planning variables using clustering over step representations, and its probing analysis has used pre-trained sentence encoders to study whether the induced planning categories are learnable and distinct. It has clustered reasoning steps to create auxiliary labels that precede explicit CoT. Su et al. (2025) have used generated latent discrete tokens to randomly replace the initial portion of reasoning traces, showing that a discrete latent vocabulary can be learned. Compared with these methods, CIRF fully segments explicit CoT into semantically grounded units and introduces a carefully designed alignment method, leading to substantial improvements in reasoning efficiency.

3 Methodology

The proposed CIRF framework converts an explicit CoT rationale into a short sequence of discrete functional tokens and then fine-tunes a language model to autoregressively generate three components: discrete functional tokens, their corresponding results, and the final answer. In this section, we formulate the problem, and describe functional token assignment, alignment, and compression.

3.1 Problem Formulation

We define a functional unit in CoT reasoning. Given a question x_i, a typical LLM can generate an answer y_i together with a rationale r_i as follows:

$$(r_i, y_i) = G(x_i; p), \tag{1}$$

where p is a prompt that elicits rationale generation. In standard CoT prompting, r_i is a reasoning trace composed of multiple textual reasoning steps. This trace can be represented as

$$r_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,m_i}), \tag{2}$$

where $s_{i,j}$ denotes the textual description of the j-th reasoning step for question x_i, and m_i is the total number of reasoning steps. We regard $s_{i,j}$ as a description of a function that produces a new result $u_{i,j}$, given contextual information including the question, prompt, and previous results $(x_i, p, u_{i,<j})$ as inputs. We then convert the textual output of explicit CoT into a sequence of functional tokens $f_{i,j}$ and their corresponding result tokens $u_{i,j}$ as follows:

$$(r_i, y_i) \rightarrow (f_{i,1}, u_{i,1}, f_{i,2}, u_{i,2}, \ldots, f_{i,m_i}, y_i). \tag{3}$$

Our abstract formulation allows variable numbers of functional and result tokens for each reasoning step. We assign new tokens to the functions while using existing textual tokens for the results. Our goal is to construct a set of functional tokens whose latent representations are reusable across a wide range of questions, enabling an LLM to accurately induce the results through implicit reasoning.

3.2 Functional Token Assignment

We first segment r_i into semantically coherent reasoning units. In our preprocessing pipeline, segmentation is primarily delimiter-based: we detect explicit ordinal markers such as "Step k," "k.," or equivalent numbered boundaries. We treat the text span between two consecutive boundaries as one segment. For each reasoning segment $s_{i,j}$, we compute a semantic representation using a pre-trained sentence embedding model G_{sent}:

$$z_{i,j} = G_{\text{sent}}(s_{i,j}) \in \mathbb{R}^{d_s}. \tag{4}$$

We use a sentence embedding model because such models are typically trained to produce semantically meaningful text representations suitable for similarity comparison, retrieval, and clustering. However, these representations are often biased toward the theme or situation of a given question, which makes it difficult to directly identify their functional types. Since the reasoning segments of a single question typically share a common situational bias, we reduce this bias by mean-centering the segment representations for each question:

$$z'_{i,j} = z_{i,j} - \frac{1}{m_i} \sum_{k=1}^{m_i} z_{i,k}. \tag{5}$$

We empirically observe that this straightforward method effectively reduces situational bias while preserving functional information. We use the centered segment embeddings as the functional representations of reasoning segments.

We use VQ-VAE (van den Oord et al., 2017) to assign functional tokens to the segments. Let $\mathcal{Z} =$

$\{x_n\}_{n=1}^{M}$ denote the collection of all encoded functional representations in the training split, where $x_n = F_{\text{enc}}(z'_{i,j}) \in \mathbb{R}^{d_e}$ and $M = \sum_i m_i$. Note that we set d_e to the target LLM embedding dimension and use a two-layer feedforward network with tanh activation for $F_{\text{enc}}(\cdot)$. We then initialize the codebook for vector quantization using a balanced clustering procedure based on the Sinkhorn-Knopp algorithm (Knight, 2008). Specifically, we first choose K provisional anchors $\{\tilde{e}_k\}_{k=1}^{K}$ from $\mathcal{Z}$, and construct a positive affinity matrix:

$$A \in \mathbb{R}_{>0}^{M\times K}, A_{n,k} = \exp\left(-\frac{\|x_n - \tilde{e}_k\|_2^2}{\lambda}\right), \quad (6)$$

where $\lambda > 0$ is a temperature parameter that controls the sharpness of the affinity distribution. We then alternately rescale the rows and columns of $\mathbf{A}$ until the resulting matrix $\mathbf{Q} \in \mathbb{R}_{>0}^{M\times K}$ lies on the doubly stochastic polytope:

$$\textstyle\sum_k Q_{n,k} = 1, \qquad \sum_n Q_{n,k} = M/K. \quad (7)$$

This procedure ensures that each anchor receives an equal share of assignment mass by construction. After obtaining the balanced soft assignment matrix, we convert it into a hard code assignment:

$$a_n = \arg\max_{k\in\{1,\ldots,K\}} Q_{n,k}. \quad (8)$$

Equivalently, for each segment $s_{i,j}$, we write

$$a_{i,j} = \arg\max_{k\in\{1,\ldots,K\}} Q_{(i,j),k}, \quad (9)$$

where $Q_{(i,j),k}$ denotes the assignment probability of segment $s_{i,j}$ to code k. Let $q_{i,j} = e_{a_{i,j}}$ denote the selected code vector for segment $s_{i,j}$. The codebook is then learned using the VQ-VAE objective:

$$\mathcal{L}_{\text{vq}} = \frac{1}{\sum_i m_i}\sum_{i=1}^{N}\sum_{j=1}^{m_i}\Big[\|F_{\text{dec}}(q_{i,j}) - z'_{i,j}\|_2^2 + \|\,\text{sg}[x_{i,j}] - q_{i,j}\|_2^2 + \beta\|x_{i,j} - \text{sg}[q_{i,j}]\|_2^2\Big], \quad (10)$$

where F_{dec} is another feedforward network, $\text{sg}[\cdot]$ is the stop-gradient operator, and β is the commitment coefficient. We use the same architecture for F_{enc} and F_{dec}. The first term is a reconstruction error, which optimizes F_{dec} to reconstruct the original centered representations from the functional code vectors. The second term updates the codebook vectors toward their assigned encoded representations, while the third term updates F_{enc} to encourage the encoded representations to remain close to their quantized code vectors. We iteratively update the codebook vectors and reassign the codes according to the procedures described above.

After quantization, each segment $s_{i,j}$ is replaced by a discrete functional token $\tau_{a_{i,j}}$. Thus, a rationale r_i can be converted into a functional token sequence $(\tau_{a_{i,1}}, \tau_{a_{i,2}}, \ldots, \tau_{a_{i,m_i}})$. To integrate these codes into the target LLM, we extend its vocabulary with K new special tokens $\{\tau_1, \ldots, \tau_K\}$, together with dedicated tokens indicating the start and end of the reasoning span $\tau_{\text{s}}, \tau_{\text{e}}$.

Finally, we initialize the new token embeddings using the rescaled codebook vectors to transfer their semantic and geometry structure:

$$e_k \leftarrow \alpha \cdot \frac{e_k}{\|e_k\|_2}, \quad (11)$$

where e_k denotes a codebook vector and α is a scaling parameter fixed to 0.01. This initialization allows the language model to exploit geometric relationships among functional codes from the beginning of training, leading to a better initial understanding of their functional roles.

3.3 Result Collection and Alignment

We introduce optional result tokens in latent reasoning, which denote the output of functional operations. The purpose of these result tokens is to provide the model with a minimal textual bridge between latent reasoning and answer generation. For each training example, we construct a result target u_i by prompting an LLM under three constraints: (1) the model should generate the result of a reasoning segment, rather than its intermediate derivation; (2) the model should generate only the result used in subsequent reasoning steps; and (3) the model should describe the result as concisely as possible. We denote this process as

$$u_i = G(x_i, r_i, y_i; p_{\text{result}}), \quad (12)$$

where p_{result} is a prompt that instructs the model to produce the results. The result u_i can be decomposed into step result $(u_{i,1}, u_{i,2}, \ldots, u_{i,m_i})$ and empty step results are allowed.

We aim to minimize textual tokens used for the results for two reasons. First, these tokens incur additional generation cost, which degrades inference efficiency. Second, we observe that exposing models to natural language reasoning traces can cause the latent token representations to collapse, as the

models tend to favor reasoning in natural language over reasoning with unseen functional tokens.

For each example, the final supervision target is a token sequence:

$$T_i = [\tau_{\text{sof}}] \oplus \mathcal{H}(F_{i,1}, ..., F_{i,m_i}) \oplus [\tau_{\text{eof}}, y_i]$$

$$s.t. \quad F_{i,j} = (\tau_{a_{i,j}}, u_{i,j}) \tag{13}$$

Here, $\oplus$ denotes concatenation and $\mathcal{H}$ is a token sequence construction function. The target LLM M_θ is then fine-tuned using the standard autoregressive language modeling objective:

$$\mathcal{L}_{\text{LM}} = -\sum_{i=1}^{N} \sum_{t=1}^{|T_i|} \log p_\theta(T_{i,t} \mid x_i, T_{i,<t}). \tag{14}$$

At inference time, CIRF requires only the fine-tuned target model M_θ. Given a question, the model generates the functional tokens and their optional result tokens, and subsequently predicts the final answer.

3.4 Result Compression

In many examples (>70%), we observe that functional token sequences alone are sufficient to generate correct answers. We therefore introduce a result compression procedure that retains only loss-reducing result units. Given the full set of result units U_i, we construct a compressed result set C_i through iterative pruning. Starting from $C_i^{(0)} = U_i$, we use a trained CIRF model as a scoring model to estimate the usefulness of each candidate.

Given $C_i^{(r)}$ at iteration r, we greedily identify the candidate unit whose removal results in the smallest increase in loss. If the best candidate increases the loss by more than a predefined threshold γ, we terminate the search. This greedy pruning strategy provides an explicit mechanism for controlling the accuracy-latency trade-off through the threshold γ: a larger γ yields shorter tokens with lower decoding cost, whereas a smaller γ retains more result units when they improve answer likelihood.

4 Experiments

In this section, we evaluate and compare CIRF with state-of-the-art efficient reasoning methods to demonstrate that it achieves an optimal accuracy-latency trade-off in latent reasoning.

4.1 Experimental Setup

Tasks and datasets. We evaluate CIRF on mathematical, symbolic, logical, and textual reasoning benchmarks. The mathematical benchmarks include GSM8K (Cobbe et al., 2021), SVAMP (Patel et al., 2021), MultiArith (Roy and Roth, 2015), and MATH-500 (Hendrycks et al., 2021). The symbolic and logical benchmarks include Coin Flip (Wei et al., 2022) and BIG-Bench Hard (Suzgun et al., 2023). The textual reasoning benchmarks include CommonsenseQA (Talmor et al., 2019), StrategyQA (Geva et al., 2021), and ScienceQA (Lu et al., 2022). GSM8K, SVAMP, MultiArith, Coin Flip, and CommonsenseQA are used as in-domain tasks for supervised fine-tuning and evaluation, whereas MATH-500, BIG-Bench Hard, StrategyQA, and ScienceQA are used as out-of-domain benchmarks. We provide further details on the evaluation protocol in Appendix A.1.

Baselines. We compare CIRF with four categories of baselines. First, Direct Answer prompts the model to generate only the final answer, whereas SFT-CoT fine-tunes the model to generate the full explicit rationale followed by the answer. Second, the compressed explicit CoT baseline, i.e., Pause Tokens (Goyal et al., 2024), reduces visible reasoning cost while remaining in natural language or repeated token space. Third, implicit CoT baselines, including iCoT-SI (Deng et al., 2024), Coconut (Hao et al., 2025), CODI (Shen et al., 2025), and SIM-CoT (Wei et al., 2025), internalize reasoning through hidden states, continuous thought representations, or step-level latent supervision. Finally, soft token baselines, including SoftCoT (Xu et al., 2025b) and SemCoT (He et al., 2025), generate compact reasoning surrogates using additional modules. We implement, train, and evaluate all baselines under controlled settings. The baseline settings are described in Appendix A.3.

4.2 Main Results

Figure 2 compares accuracy and inference time across in-domain and out-of-domain settings. CIRF consistently lies in a favorable region of the Pareto frontier. $\text{CIRF}_{\text{Full}}$ achieves the highest accuracy among the CIRF variants, while $\text{CIRF}_{\text{Fast}}$ and $\text{CIRF}_{\text{Faster}}$ substantially reduce inference time by compressing result units.

Specifically, with Qwen3-8B, all CIRF variants outperform the state-of-the-art baseline CODI in terms of accuracy. With a small additional

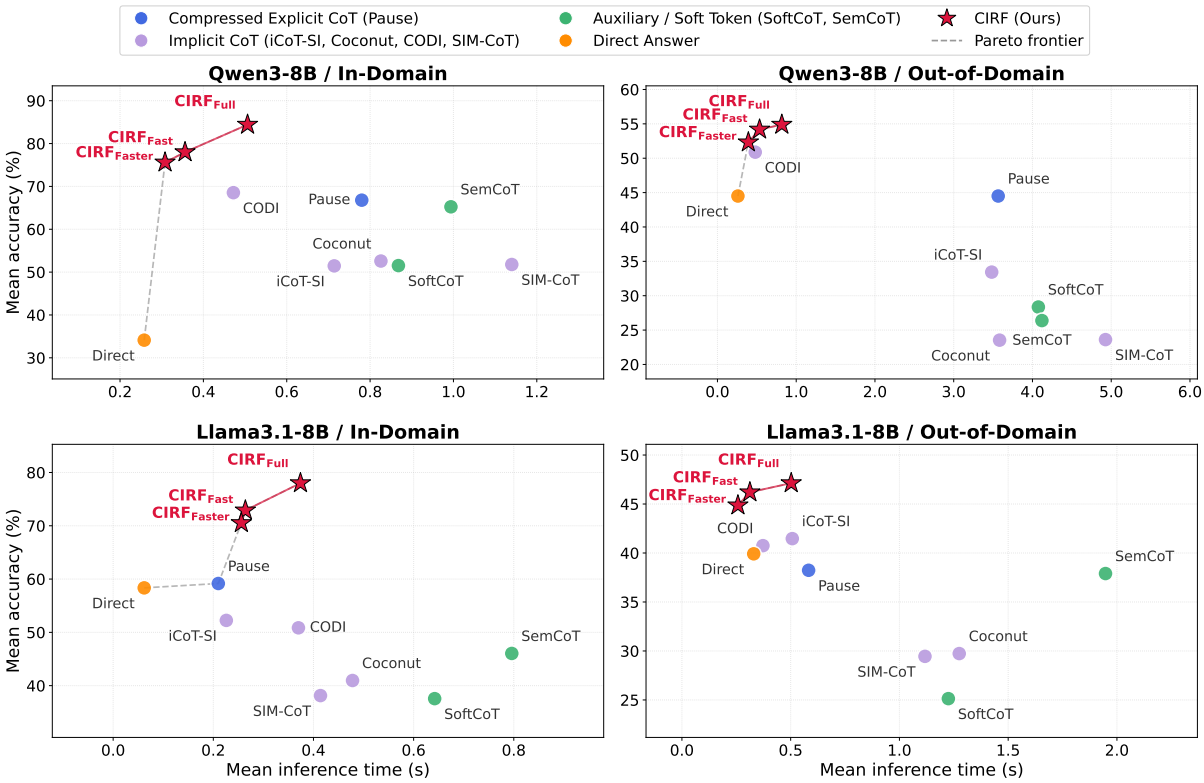

Figure 2: Accuracy-latency trade-off across in-domain and out-of-domain benchmarks. Each point reports the mean accuracy and mean inference time of a reasoning method. Task-wise full results are reported in Appendix B.

latency, $\text{CIRF}_{\text{Full}}$ achieves higher accuracy than CODI, with a gap of up to 45.7%p in GSM8K ($\text{CIRF}_{\text{Full}}$ 71.2% vs. CODI 25.5%). $\text{CIRF}_{\text{Fast}}$ and $\text{CIRF}_{\text{Faster}}$ exhibit better accuracy and latency than CODI, while $\text{CIRF}_{\text{Faster}}$ provides stronger acceleration than $\text{CIRF}_{\text{Fast}}$ at the cost of accuracy. In our experiments, compressing result units causes the largest accuracy degradation on Coin Flip (-20.9%p for $\text{CIRF}_{\text{Fast}}$ and -20.7%p for $\text{CIRF}_{\text{Faster}}$) and GSM8K (-6.5%p for $\text{CIRF}_{\text{Fast}}$ and -16.5%p for $\text{CIRF}_{\text{Faster}}$), which inherently require multiple reasoning steps and intermediate state tracking. Other baselines, such as Pause Tokens and soft token methods, show worse accuracy-latency trade-offs, possibly due to the limited reasoning capacity of a single placeholder token and the additional latency incurred by inference with multiple models.

We extend the Qwen3-8B experiments to out-of-domain tasks. CIRF achieves the most favorable accuracy-latency trade-offs, followed by CODI. CIRF consistently exhibits stable reasoning behavior, whereas most baselines produce degenerate results and long latencies on MATH-500, Big-Bench Hard, and StrategyQA, which also require multi-hop reasoning and state tracking. We further conduct experiments with another model, Llama3.1-8B, and observe consistent trends. CIRF achieves the Pareto frontier, with faster inferences at slightly lower accuracy. A notable difference in this experiment is the improvement of Pause Tokens, which suggests that Llama3.1-8B may better utilize placeholder tokens for latent reasoning. These results demonstrate the generalizability of CIRF across out-of-domain tasks and multiple models.

An interesting pattern is that all in-domain CIRF results except for $\text{CIRF}_{\text{Faster}}$ with Llama3.1-8B form a linear trend in the accuracy-latency plot. Based on this trend, we hypothesize that a scaling law may exist for latent reasoning with implicit tokens. We leave a more detailed examination of this pattern to future work.

4.3 Analyses

We conduct in-depth analyses to better understand the performance improvements. Unless otherwise specified, we use $\text{CIRF}_{\text{Full}}$ with Qwen3-8B.

Codebook size. In Figure 3, we vary the number of functional tokens K to study the granularity of the functional vocabulary. A very small codebook may merge distinct reasoning operations, whereas a very large codebook may fragment semantically

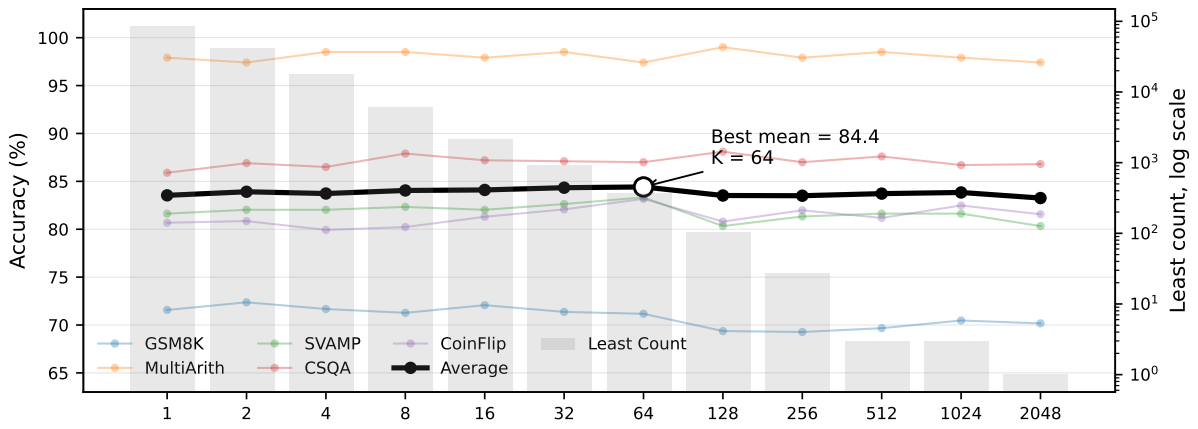

Figure 3: Effect of codebook size on downstream accuracy and post-training code utilization. The solid line reports average accuracy across benchmarks, the task-specific curves show per-dataset accuracy, and the gray bars indicate the count of the least used code on a logarithmic scale. Delta accuracy comparisons are reported in Appendix B.

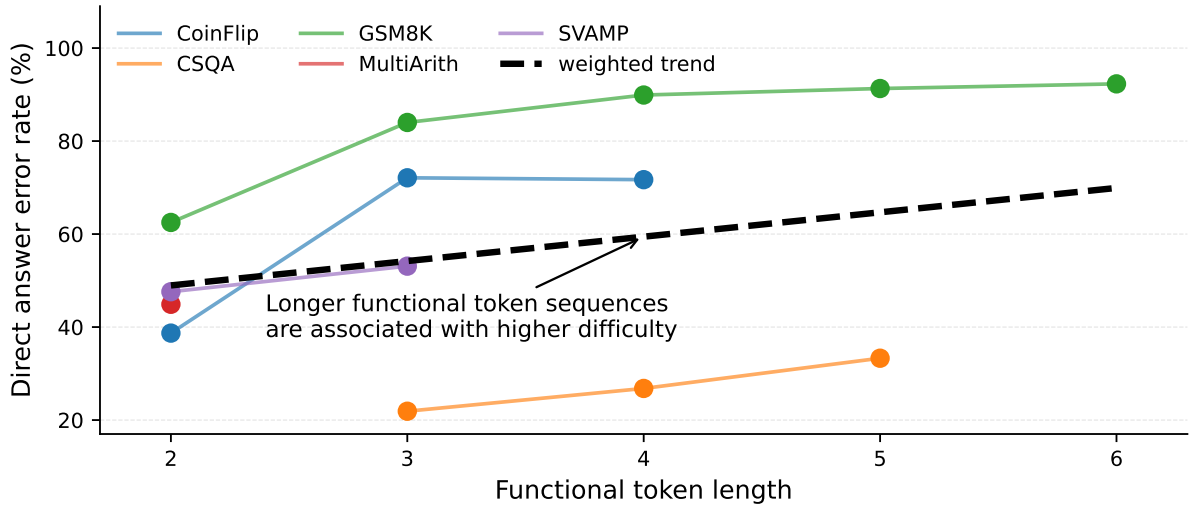

Figure 4: Relationship between generated functional token length and instance difficulty. Examples are grouped by the number of generated functional tokens, and the error rate is reported for each group.

similar operations and reduce token reuse. The codebook size ablation shows that moderate values of K provide the best performance. Average performance gradually improves as the number of functional tokens increases, until it reaches an optimal value. Performance then decreases when the codebook becomes too large, suggesting that the functional vocabulary should be sized considering both the diversity and token frequency.

Adaptive reasoning. CIRF generates a variable-length sequence of functional tokens. As shown in Figure 4, we analyze whether this length reflects reasoning difficulty by grouping examples according to the number of generated functional tokens and measuring the corresponding error rate under direct answering. The results show a positive trend: examples requiring longer functional token sequences tend to have higher direct answering error rates. This trend indicates that CIRF performs latent reasoning adaptively with respect to instance difficulty and complexity, dynamically adjusting its reasoning length through the number of generated functional tokens.

Functional token interpretability. To examine whether functional tokens correspond to reusable reasoning operations, we retrieve reasoning segments assigned to frequent codes and inspect their shared semantics in Table 1. The assigned segments reveal coherent operation types, such as answer selection, commonsense reasoning, addition, subtraction, and multiplication/division. This qualitative evidence suggests that the learned codes are

Code	Frequency	Operation	Assigned CoT Segments
25	3,528	Choose answer	... the best answer is (E) use television. This is because the use of a remote ... ... Based on the analysis, the most logical and relevant answer is (C) suffering pain, ... the best option is (C) dribble, as it accurately describes what happens to ...
18	3,335	Commonsense	... an attic is typically above the main living space of a house ... ... A beach is an outdoor location, often very bright due to the sun. ... ... Army captains might participate in recreational activities such as basketball, ...
23	3,051	Mul/Div	4 bananas * $1/banana = $4 ... She adds twice that amount: 3 statues * 2 = 6 statues. ... ... so she tried on 16 * 2 = 32 pairs.
16	2,745	Addition	... $40 (coat) + $30 (shoes) = $70 ... Her total savings are $50 + $15 + $30 = $95. ... Each basket has 4 red peaches + 6 green peaches = 10 peaches.
3	2,635	Subtraction	... The wallet costs $100, and Betty has $95. She needs $100 - $95 = $5 more ... ... Amount for insurance: $5200 - $4940 = $260 ... ... 28 - 15 = 13

Table 1: Qualitative examples of learned functional token assignments. For each code, we report its frequency, the interpreted operation type, and three CoT segments assigned to the code.

Methods	Bias ↓	Avg Cos. ↓	Max Cos. ↓
Representation			
`random`	0.128	0.000	0.109
`raw`	0.405	0.164	0.955
`q-cent.`	0.497	0.246	0.974
`mean-cent.`	0.124	0.009	0.756
Codebook construction			
K-means	0.131	0.011	0.869
Sinkhorn	0.124	0.009	0.756
Codebook size			
$K = 32$	0.124	0.009	0.756
$K = 128$	0.131	0.010	0.842
$K = 512$	0.110	0.010	0.919
$K = 2048$	0.085	0.007	0.939

Table 2: Functional token geometry under various settings. Bias denotes the bias share, defined as $\|\mu\|/\mathbb{E}[\|w\|]$, where μ is the mean vector of the LLM functional token embeddings and w is an individual token vector. Max/Avg Cos. denotes the maximum/average pairwise cosine similarity.

neither arbitrary identifiers nor surface-level syntactic patterns, but instead capture reusable functional operations that recur across examples and datasets in an interpretable manner.

Functional token geometry. We further analyze the geometry of the functional token embeddings. This analysis examines whether the learned functional embeddings are geometrically diverse rather than collapsed into a single vector. Table 2 shows four main observations. First, raw segment embeddings and question-centered embeddings exhibit large bias shares and high maximum cosine similarities in the final language model embedding space. This indicates that their token vectors are strongly affected by shared contextual offsets, which can reduce the separability of reusable functional operations. Second, the proposed Sinkhorn-initialized functional codebook maintains a low bias share, suggesting that balanced clustering effectively prevents collapse into a shared direction. Third, randomly initialized embeddings can appear geometrically well spread, as indicated by low cosine similarity and a low bias share, but such geometry is insufficient for functional reasoning because it does not preserve the semantic structure extracted from CoT segments. Finally, as the codebook size increases, the maximum cosine similarity also increases, indicating that overly large codebooks may introduce redundant functional tokens.

5 Conclusion

We presented CIRF, an implicit chain-of-thought framework that converts explicit reasoning traces into reusable discrete functional tokens. Through carefully designed functional token construction methods, CIRF generates dynamic sequences of functional tokens using a single autoregressive language model. Experiments across in-domain and out-of-domain reasoning benchmarks show that CIRF achieves a favorable accuracy-latency trade-off compared with state-of-the-art latent reasoning baselines, benefiting from its preservation of functional operational structure. We believe that CIRF offers a promising direction for building language models that reason more efficiently across a wide range of complex tasks.

Limitations

While we have demonstrated the frontier-level efficiency of CIRF, some limitations open promising avenues for future research. First, the quality of functional token supervision may depend on the quality and structure of the explicit CoT traces. We currently rely on explicit ordinal markers to segment rationales into reasoning units. More robust segmentation methods could further improve the reliability of the functional unit extraction process. In addition, the result token alignment and compression procedure introduces an additional design trade-off. An optimization method for this procedure may help identify a better accuracy-latency balance. Finally, as our experiments mainly focus on 8B-scale models and several tasks, the behavior of CIRF on differently sized models and additional tasks like long-context and tool-augmented reasoning would be explored in the future.

Ethics Statements

CIRF is a general purpose framework for improving the efficiency of chain-of-thought reasoning in large language models. The method does not introduce new specific capabilities for harmful domains, nor does it rely on private, sensitive, or user-identifying data. The primary potential risk is indirect: because CIRF performs part of the reasoning process through discrete functional tokens rather than fully explicit natural-language rationales, it may reduce the human interpretability of individual reasoning steps compared with fully explicit CoT. However, CIRF partially mitigates this concern by constructing functional tokens from explicit CoT segments and by providing analyses showing that the learned tokens correspond to interpretable reasoning operations.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Yuntian Deng, Yejin Choi, and Stuart Shieber. 2024. From explicit CoT to implicit CoT: Learning to internalize CoT step by step. *CoRR*, abs/2405.14838.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *CoRR*, abs/2407.21783.

Gemma Team. 2025. Gemma 3 technical report. *CoRR*, abs/2503.19786.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2024. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, Vienna, Austria. OpenReview.net.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2025. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling (COLM)*, Montreal, Canada. OpenReview.net.

Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. 2025. Don't overthink it. preferring shorter thinking chains for improved LLM reasoning. *CoRR*, abs/2505.17813.

Yinhan He, Wendy Zheng, Yaochen Zhu, Zaiyi Zheng, Lin Su, Sriram Vasudevan, Qi Guo, Liangjie Hong, and Jundong Li. 2025. SemCoT: Accelerating chain-of-thought reasoning through semantically-aligned implicit tokens. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS 2025)*, San Diego, CA, USA. OpenReview.net.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Philip A. Knight. 2008. The sinkhorn–knopp algorithm: Convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275.

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

Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *Advances in Neural Information Processing Systems*, volume 35, pages 2507–2521. Curran Associates, Inc.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Jacob Pfau, William Merrill, and Samuel R. Bowman. 2024. Let's think dot by dot: Hidden computation in transformer language models. *CoRR*, abs/2404.15758.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. 2025. CODI: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 677–693, Suzhou, China. Association for Computational Linguistics.

Zayne Rea Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. 2025. To CoT or not to CoT? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*, Singapore. OpenReview.net.

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. 2025. Token assorted: Mixing latent and text tokens for improved language model reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 57144–57163. PMLR.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

A"aron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30, pages 6306–6315. Curran Associates, Inc.

Xinyi Wang, Lucas Caccia, Oleksiy Ostapenko, Xingdi Yuan, William Yang Wang, and Alessandro Sordoni. 2024. Guiding language model reasoning with planning tokens. In *First Conference on Language Modeling (COLM)*. OpenReview.net.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. 2025. SIM-CoT: Supervised implicit chain-of-thought. *arXiv preprint arXiv:2509.20317*.

Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2026. When more is less: Understanding chain-of-thought length in LLMs. In *The Fourteenth International Conference on Learning Representations*. OpenReview.net.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. TokenSkip: Controllable chain-of-thought compression in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3351–3363, Suzhou, China. Association for Computational Linguistics.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025a. Chain of draft: Thinking faster by writing less. *CoRR*, abs/2502.18600.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025b. SoftCoT: Soft chain-of-thought for efficient reasoning with LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23336–23351, Vienna, Austria. Association for Computational Linguistics.

A Detailed Experimental Settings

A.1 Tasks and Datasets

Mathematical reasoning. We include multi-step arithmetic benchmarks, where explicit CoT has been consistently effective:

- **GSM8K** (Cobbe et al., 2021): grade-school math word problems requiring multi-step arithmetic reasoning.
- **SVAMP** (Patel et al., 2021): adversarially constructed arithmetic word problems designed to test robustness to variations in problem structure.
- **MultiArith** (Roy and Roth, 2015): multi-step arithmetic problems with relatively short problem statements.
- **MATH-500** (Hendrycks et al., 2021): competition-level mathematical reasoning problems. We use this dataset primarily as a harder out-of-distribution benchmark.

Symbolic and logical reasoning. To evaluate whether CIRF can represent reusable operations beyond arithmetic, we include symbolic and logical reasoning tasks:

- **Coin Flip** (Wei et al., 2022): a state-tracking task requiring updates over a sequence of operations.
- **BIG-Bench Hard** (Suzgun et al., 2023): a wide range of tasks with each designed to evaluate different aspects of logical reasoning.

Commonsense and multi-hop reasoning. To examine whether CIRF is useful on semantic reasoning steps, we include commonsense and multi-hop question answering datasets:

- **CommonsenseQA** (Talmor et al., 2019): multiple-choice commonsense reasoning.
- **StrategyQA** (Geva et al., 2021): implicit multi-hop reasoning requiring decomposition into subquestions.
- **ScienceQA** (Lu et al., 2022): science-domain multiple-choice reasoning. We use the text-only subset unless otherwise specified.

A.2 CIRF Settings

CoT Collection We collect an explicit rationale using Gemma 3 27B (Gemma Team, 2025) and Llama 3.1 70B (Dubey et al., 2024) on the training sets of GSM8K, SVAMP, MultiArith, CommonsenseQA, and Coin Flip for the base setting. We generate CoTs using a simple prompt of *"Solve the following problem step by step:{question}"*. The rationale is normalized into a step-structured format before segmentation. We set the decoding temperature, top-p, and maximum generation length to 0.7, 0.95, and 4096, respectively. We filter out the entries that are not matched with our segmentation format, leading to elimination of 0.8% of all generated entries.

Functional Token Construction We encode each segmented reasoning unit using the Qwen3-8B embedding model, and then construct a VQ codebook with the number of functional codes selected from $\{32, 64, 128, 256\}$ based on the validation accuracy, and the Sinkhorn temperature of 0.05. The dimension of code embedding vectors is identical to the embedding dimension of the reasoning model. We train the codebook for 10 epochs with the commitment coefficient of 1.0. The resulting code IDs are added to the backbone tokenizer as special tokens, and their embeddings are initialized by projecting the learned codebook vectors into the model embedding space.

Supervised Fine-Tuning Unless otherwise mentioned, we use AdamW with learning rate 2×10^{-5}, weight decay 0.1, and batch size 32. We train for 2 epochs and select the final checkpoint. We use cosine learning rate scheduling with 100 warmup iterations. We train models on NVIDIA RTX PRO 6000 GPUs with BF16 precision, taking approximately 1 GPU hour on the base settings.

Inference Settings We use nucleus sampling with top-p 0.95 and temperature 0.7. We provide one example and measure its correctness and latency, averaged over all examples. We execute three runs and report the average performance. For mathematical and symbolic reasoning tasks, we use exact-match accuracy after answer normalization. For multiple-choice tasks, we evaluate whether the predicted option matches the ground truth. We report end-to-end time from input to final answer completion as an efficiency metric.

Dataset	Domain	Train	Test	Avg. CoT Tokens	Avg. Func. Tokens
In-Domain Tasks					
GSM8K	Math	7,473	1,319	194.8	3.95
SVAMP	Math	700	300	131.6	3.10
MultiArith	Math	420	180	130.3	3.56
Coin Flip	Symbolic	20,000	3,333	133.7	3.00
CommonsenseQA	Commonsense	9,741	1,221	294.6	8.19
Out-of-Domain Tasks					
MATH-500	Math	–	500	–	–
BIG-Bench Hard	Symbolic	–	6,511	–	–
StrategyQA	Multi-hop	–	229	–	–
ScienceQA	Science	–	4,241	–	–

Table 3: Dataset statistics for in-domain and out-of-domain evaluation. In-domain tasks are used for supervised fine-tuning and evaluation, whereas out-of-domain tasks are used only for evaluation. Average CoT tokens and average functional tokens are measured on Qwen3-8B.

Backbone models. We evaluate CIRF on multiple backbone families to verify that its effectiveness is not specific to a single model architecture. Unless otherwise stated, all backbone models are instruction-tuned causal language models, i.e., Llama3.1-8B and Qwen3-8B.

CIRF Variants We evaluate the following variants of CIRF to characterize the trade-off between reasoning quality and inference cost:

- **CIRF**$_{\text{Full}}$: generates functional tokens and all aligned result units.
- **CIRF**$_{\text{Fast}}$: generates functional tokens and compressed result units selected by our loss-reduction criterion, whose threshold is set to 0.1.
- **CIRF**$_{\text{Faster}}$: generates functional tokens and more aggressively pruned result units with a threshold set to 0.2.

We list the hyper-parameter settings in Table 7.

A.3 Baselines

We compare CIRF with baselines from four categories. We implement the baselines based on their publicly released codes for scientific purpose, complying with their license and intended use. All baselines are trained on a single RTX PRO 6000 GPU with bfloat16 using the AdamW optimizer, unless otherwise noted. For the Math500 dataset, we set the maximum output length to 512. Evaluation is performed at temperature 0.7, averaged over three inference runs from a trained model. Most fine-tuned methods use LoRA (Hu et al., 2022) with r=128 and α=32, with Baseline-specific settings described below.

Direct and Explicit CoT Baselines These baselines represent the standard explicit CoT settings, which decode natural language rationales.

- **Direct Answer**: the model is prompted to produce only the final answer. This baseline measures the performance of non-rationale generation.
- **SFT-CoT**: the backbone is supervised fine-tuned to generate the full explicit rationale followed by the final answer.

Compressed Explicit CoT Baselines These methods remain in natural-language token space. They are important baselines because CIRF also reduces visible reasoning length, but does so by replacing reasoning operations with discrete functional tokens.

- **Chain-of-Draft (CoD)** (Xu et al., 2025a): generates concise intermediate reasoning drafts instead of verbose CoT traces. Following the official implementation,[1] CoD is applied at inference time through prompting alone and requires no additional training.
- **TokenSkip** (Xia et al., 2025): selectively skips less important reasoning tokens to reduce CoT length. We adopt the authors' implementation[2] and fine-tune for 10 epochs with a learning rate of 1e−4 and weight decay of 0.01.
- **Pause Tokens** (Goyal et al., 2024): inserts repeated special tokens before answer generation to provide additional computation steps.

[1] https://github.com/xu3kev/chain-of-draft
[2] https://github.com/hemingkx/TokenSkip

We insert five learnable <pause> tokens before the answer and fine-tune for 10 epochs with a learning rate of $1e{-4}$ and weight decay of 0.01, following the implementation released with SemCoT.[3]

Implicit CoT Baselines These methods are closest to CIRF in objective, as they aim to reduce or remove explicit natural-language rationales. However, they differ in whether the reasoning carrier is continuous, homogeneous, externally decoded, or semantically discrete.

- **iCoT-SI** (Deng et al., 2024): distills explicit CoT supervision into hidden representations. We progressively remove the rationale during training, dropping eight tokens per epoch over 10 epochs with a learning rate of $5e{-5}$, following the implementation released with SemCoT.[A.3]

- **Coconut** (Hao et al., 2025): uses the previous hidden state as a continuous thought representation and feeds it back as the next input embedding. We use five continuous thought tokens learned through a three-stage curriculum and train for 10 epochs with a learning rate of $1e{-5}$ and weight decay of 0.01, following the implementation released with SemCoT.[A.3]

- **CODI** (Shen et al., 2025): aligns explicit and implicit CoT through self-distillation. We adopt the official implementation[4] with six latent tokens, training for 5 epochs with a learning rate of $8e{-4}$, weight decay of 0.1, a distillation factor of $\gamma{=}20$, an effective batch size of 128, and a cosine schedule with warmup ratio 0.03.

- **SIM-CoT** (Wei et al., 2025): uses explicit step-level supervision to stabilize and diagnose latent reasoning states. Building on Coconut, we add auxiliary-decoder supervision with five thought tokens and train for 10 epochs with a learning rate of $1e{-5}$, weight decay of 0.01, three curriculum stages, and an auxiliary loss weight of 1.0, following the official implementation.[5]

Auxiliary Model and Soft Token Baselines These baselines are essential for evaluating wall-clock efficiency, because auxiliary modules can reduce the number of decoded tokens but may introduce additional prompt-dependent computation.

- **SoftCoT** (Xu et al., 2025b): uses a lightweight assistant model to generate instance-specific soft thought tokens for the main LLM. We train only the projection layer that maps the assistant's hidden states into five soft thought tokens, keeping both the assistant and the main model frozen, for 10 epochs with a learning rate of $1e{-3}$ and weight decay of 0.01, following the implementation released with SemCoT.[A.3]

- **SemCoT** (He et al., 2025): introduces semantic alignment for implicit tokens to preserve reasoning information while accelerating generation. We follow the official implementation[A.3] and use five semantic thought tokens during training and a single token at evaluation.

B Additional results

Functional representation extraction. We study how to extract functional information from reasoning-segment embeddings. We compare random embeddings (`random`), raw segment embeddings without centering (`raw`), segment embeddings centered by their corresponding question embedding (`q-cent.`), and the mean-centered embeddings (`mean-cent.`). As shown in Table 4, `mean-cent.` achieves the best average downstream accuracy, improving over `raw` and `q-cent.` These results support that mean-centering suppresses question-specific situational bias while preserving reusable reasoning functionality.

Codebook construction. We examine the effect of codebook initialization and token-embedding initialization in Table 5. We compare random initialization, K-means initialization, Sinkhorn initialization, and methods without semantic embedding initialization. The results suggest that both the balanced code assignment and trained embeddings are important for learning reusable functional token embeddings during language model fine-tuning.

Codebook size analysis. We measure the delta accuracy, which is an accuracy difference from the minimum accuracy, for each codebook size, as shown in Figure 5.

[3] https://github.com/YinhanHe123/SemCoT
[4] https://github.com/zhenyi4/codi, [A.3]
[5] https://github.com/InternLM/SIM-CoT

Method	GSM8K	MultiArith	SVAMP	CommonsenseQA	CoinFlip	Avg.
random	35.3	79.6	71.0	78.1	78.0	68.4
raw	42.1	92.9	81.3	85.4	79.3	76.2
q-cent.	69.0	96.8	82.7	87.3	81.8	83.5
mean-cent.	**71.2**	**97.4**	**83.3**	**87.0**	**83.2**	**84.4**

Table 4: Downstream accuracy of functional representation methods after codebook training and CIRF fine-tuning.

Init. Code	Init. Emb.	GSM8K	MultiArith	SVAMP	CommonsenseQA	CoinFlip	Avg.
Random	Random	35.3	79.6	71.0	78.1	78.0	68.4
Random	Trained	**71.5**	96.9	82.6	**87.1**	81.2	83.9
K-means	Trained	71.0	96.4	80.6	86.7	81.9	83.3
Sinkhorn	Trained	71.2	**97.4**	**83.3**	87.0	**83.2**	**84.4**
Sinkhorn	Random	70.8	95.2	81.3	87.1	79.0	82.7

Table 5: Codebook-construction ablation. We compare different code initialization strategies and functional-token embedding initializations.

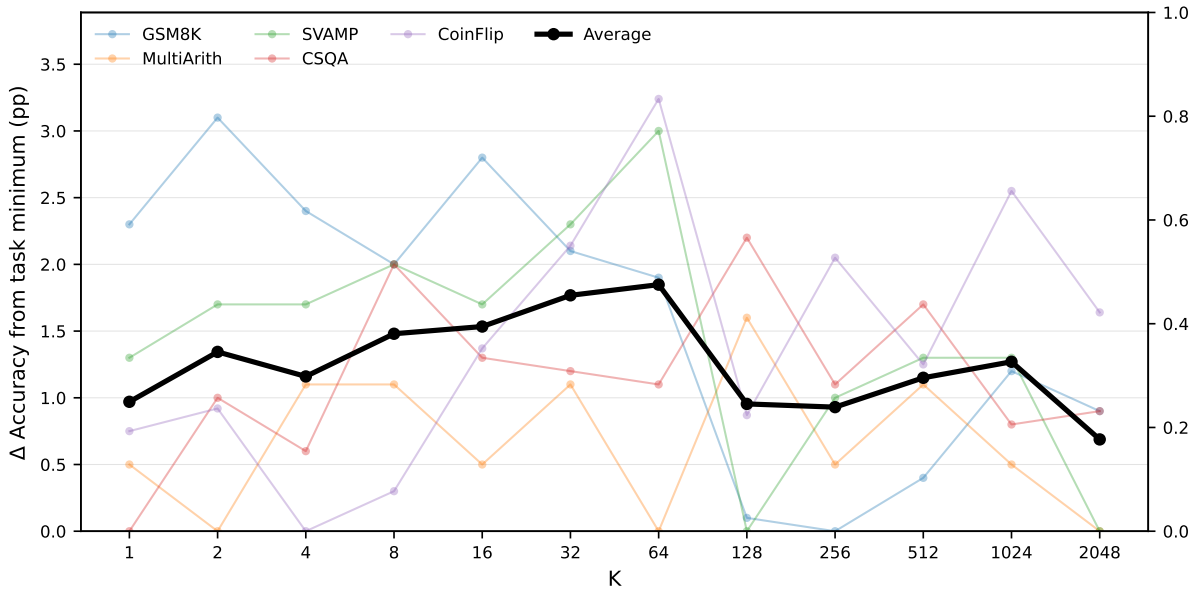

Figure 5: Effect of codebook size on downstream delta accuracy from task minimum and post-training code utilization. The solid line reports average delta accuracy across benchmarks and task-specific curves show per-dataset accuracy.

Method	Used ↑	AMI ↓	Puri. ↓	Coll. ↓	Uniq. ↑
raw	**1.000**	0.113	0.010	0.070	3.28
q-cent.	**1.000**	0.043	0.007	0.008	4.03
mean-cent.	**1.000**	**0.000**	**0.003**	**0.000**	**4.55**

Table 6: Intrinsic evaluation of functional representation methods for code assignment with $K = 128$.

Functional code analysis. We compare the clustering results of raw segment embeddings, question-centered embeddings, and the mean-centered embeddings. *Used* denotes the fraction of activated codes. *AMI* measures the dependence between assigned codes and question identity. *Puri.* reports the size-weighted cluster purity with respect to question identity. *Coll.* denotes the fraction of examples whose segments are assigned to a single code. *Uniq.* reports the mean number of distinct codes per example. We observe that the mean-centered representation substantially reduces the association between code assignments and question identity, as indicated by lower adjusted mutual information and lower cluster purity in Table 6. These results support that mean-centering suppresses question-specific situational bias while preserving reusable reasoning functionality.

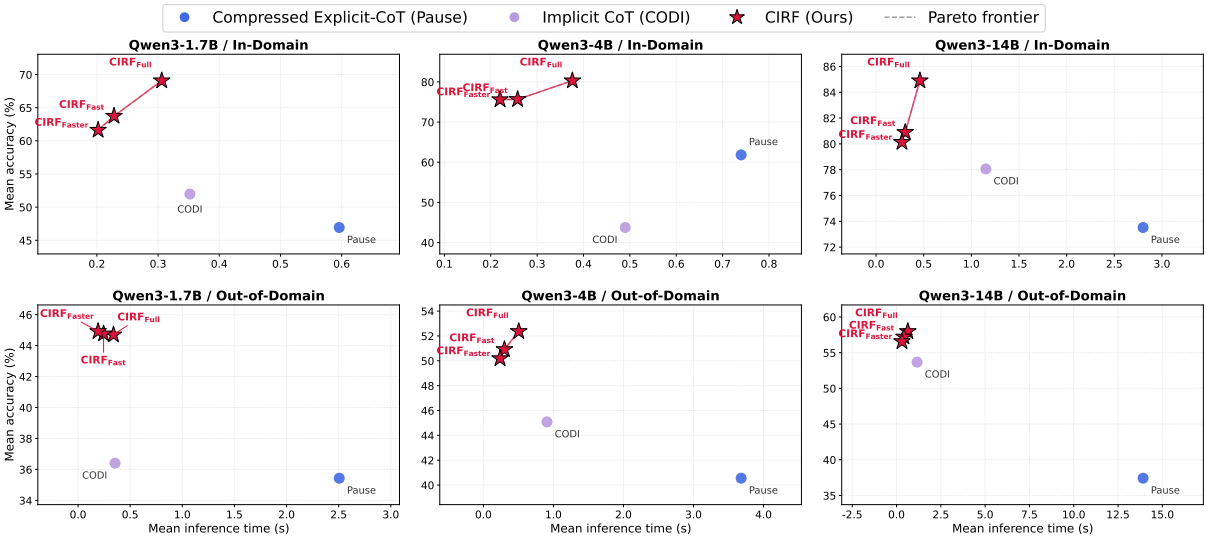

Figure 6: Scaling experiments on Qwen3-1.7B, 4B, and 14B.

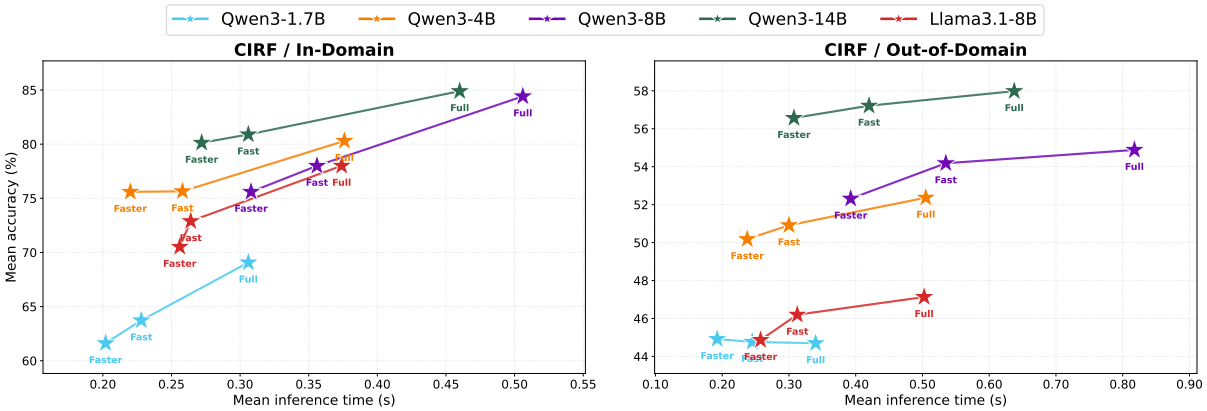

Figure 7: Accuracy vs. inference time for CIRF variants across five backbone LLMs.

Full results. We report the task-wise accuracy and latency of the experimental results in Table 8.

Effect of model scale. We extend our evaluation to additional Qwen3 scales (1.7B, 4B, and 14B). As shown in Figure 6, the Pareto optimal CIRF variants dominate both state-of-the-art baselines (CODI and Pause) at every scale, showing that the accuracy and efficiency gains generalize across model sizes.

CIRF Variant Comparison. Figure 7 compares CIRF variants aggregated across all backbone LLMs. The Fast and Faster variants substantially reduce latency while maintaining competitive accuracy. We observe that in-domain accuracies across different backbones remain similar with respect to inference time, whereas out-of-domain accuracies are different. A trend can be observed that stronger backbone models show stronger generalizability.

C Information About Use Of AI Assistants

We have used AI assistants in language editing of the paper and implementation of partial code, all of which are double-checked by humans.

Parameter	Value
Step 1: Teacher CoT generation	
Generation temperature	0.7
Generation top-p	0.95
Max generated tokens	4096
Step 2: Result alignment	
Generation temperature	0.2
Generation top-p	0.9
Max generated tokens	2048
Step 3: Codebook	
Embedding model	`Qwen3-Embedding-8B`
Codebook vector dimension	4096
MLP hidden dimension	1024
Autoencoder training epochs	30
VQ epochs	10
Codebook batch size	128
Codebook learning rate	10^{-4}
Gradient clip	1.0
VQ commitment β	1.0
Sinkhorn λ	0.05
Sinkhorn iterations	3
Codebook size K	32, 64, 128, 256
Vector scale α	0.01
Step 4: LM fine-tuning	
Hidden size	4096
Layers / heads	36 / 32
Training epochs	2
Batch size	4
Gradient accumulation steps	8
Effective batch	32
Optimizer	AdamW
Learning rate	2×10^{-5}
Weight decay	0.1
Adam β_1, β_2	0.9, 0.999
Adam ϵ	10^{-8}
LR scheduler	cosine
Warmup steps	100
Min LR ratio	0.1
Gradient clip	1.0
Step 5: Evaluation	
Generation temperature	0.7
Generation top-p	0.9
Greedy segment selection	
Loss advantage threshold γ	0, 0.1, 0.2

Table 7: Hyper-parameter settings.

	Method	Metric	*In-Domain Tasks*					*Out-of-Domain Tasks*			
			CoinFlip	**Common**	**GSM8k**	**MultiArith**	**SVAMP**	**Math-500**	**BBH**	**StrategyQA**	**ScienceQA**
Qwen3-8B	Ans (Base)	Acc (%)	34.83±1.84	77.67±0.85	17.67±1.93	11.11±1.81	29.33±0.47	16.40±0.85	37.47±0.90	49.13±1.05	75.00±0.57
		Time(s)	**0.26±0.00**	**0.25±0.00**	**0.26±0.00**	**0.26±0.00**	**0.26±0.00**	**0.26±0.00**	**0.27±0.00**	**0.25±0.00**	**0.26±0.00**
	CoT	Acc (%)	**98.67±0.62**	84.83±0.47	**90.67±1.55**	**100.00±0.00**	**91.17±0.47**	**34.93±1.73**	**48.07±1.46**	57.47±0.81	41.33±28.38
		Time(s)	4.39±0.03	7.11±0.03	4.83±0.06	3.29±0.02	3.21±0.01	9.79±0.04	7.37±0.06	9.27±0.11	6.41±0.04
	CoD	Acc (%)	64.50±0.41	19.17±0.85	6.50±0.71	32.41±1.05	16.67±2.46	27.27±0.90	7.40±0.16	3.27±0.52	18.47±0.52
		Time(s)	2.85±0.01	2.87±0.01	2.91±0.01	2.87±0.01	2.88±0.01	13.28±0.04	2.86±0.01	2.84±0.01	2.84±0.01
	TokenSkip	Acc (%)	53.00±20.36	23.50±3.54	36.00±0.41	14.07±1.05	45.50±2.45	20.80±1.14	14.33±1.51	27.33±2.78	50.13±4.06
		Time(s)	2.89±0.42	2.88±0.43	2.89±0.43	2.88±0.43	2.89±0.43	14.88±2.15	2.91±0.42	3.18±0.44	3.18±0.43
	Pause	Acc (%)	74.33±7.19	71.83±3.30	32.33±2.59	87.96±2.88	67.50±4.30	22.33±3.20	45.07±0.19	40.00±8.74	70.60±1.56
		Time(s)	0.74±0.02	0.78±0.00	0.80±0.00	0.79±0.00	0.79±0.00	12.25±0.21	0.79±0.00	0.53±0.07	0.69±0.02
	iCoT-SI	Acc (%)	25.00±5.31	78.17±5.04	24.67±1.55	69.44±9.26	60.00±7.38	12.53±2.41	40.73±2.36	7.40±2.12	73.07±0.34
		Time(s)	0.73±0.25	0.71±0.12	0.79±0.18	0.65±0.29	0.69±0.24	11.86±0.86	0.69±0.14	0.75±0.05	0.63±0.12
	Coconut	Acc (%)	48.33±3.57	57.17±7.26	27.50±2.27	67.59±0.52	62.33±1.25	27.87±2.32	11.07±2.13	3.60±1.02	51.60±4.38
		Time(s)	0.82±0.00	0.82±0.00	0.83±0.00	0.83±0.00	0.83±0.00	11.91±0.12	0.83±0.00	0.80±0.01	0.79±0.00
	CODI	Acc (%)	72.33±13.66	82.00±2.86	33.00±5.61	82.04±7.55	73.33±2.49	18.07±2.12	46.80±1.47	64.53±3.43	74.13±0.66
		Time(s)	0.45±0.00	0.45±0.00	0.50±0.01	0.47±0.00	0.49±0.00	0.54±0.03	0.47±0.02	0.46±0.02	0.45±0.00
	SIM-CoT	Acc (%)	52.50±5.02	54.83±10.27	25.67±3.79	67.22±3.60	58.67±5.20	28.93±2.45	11.47±4.22	4.13±0.25	49.93±8.07
		Time(s)	1.13±0.02	1.14±0.01	1.15±0.02	1.14±0.01	1.14±0.01	16.39±0.11	1.14±0.01	1.10±0.01	1.08±0.01
	SoftCoT	Acc (%)	60.17±2.25	58.33±6.06	21.50±2.48	64.44±6.71	53.17±1.84	13.67±3.67	30.60±5.52	7.07±2.68	62.07±5.31
		Time(s)	0.87±0.01	0.86±0.00	0.88±0.01	0.86±0.00	0.87±0.01	13.75±0.21	0.87±0.01	0.85±0.00	0.83±0.02
	SemCoT	Acc (%)	46.33±3.70	**98.17±0.85**	30.33±0.47	80.37±2.92	71.00±2.04	19.40±8.56	20.93±1.79	3.87±1.57	61.33±4.34
		Time(s)	0.99±0.00	0.99±0.00	1.01±0.00	0.99±0.00	0.99±0.00	13.58±0.63	0.99±0.00	0.96±0.01	0.95±0.01
	CIRF$_{\text{Full}}$	Acc (%)	83.17±1.55	87.00±0.41	71.17±1.70	97.41±0.52	83.33±1.03	28.27±0.84	46.67±1.09	**69.07±0.34**	75.53±1.09
		Time(s)	0.48±0.01	0.30±0.00	0.84±0.01	0.45±0.01	0.46±0.00	1.48±0.01	0.87±0.02	0.49±0.01	0.43±0.01
	CIRF$_{\text{Fast}}$	Acc (%)	62.33±0.24	86.83±0.85	64.67±0.24	93.33±0.00	82.83±0.24	27.60±0.59	44.93±0.82	67.73±0.57	**76.47±1.11**
		Time(s)	0.22±0.00	0.28±0.00	0.65±0.02	0.30±0.01	0.33±0.01	0.92±0.02	0.55±0.01	0.36±0.01	0.31±0.00
	CIRF$_{\text{Faster}}$	Acc (%)	62.50±0.00	87.00±0.41	54.67±1.31	93.33±0.79	80.50±0.00	23.87±0.75	44.60±0.43	65.67±0.96	75.13±1.09
		Time(s)	0.22±0.00	0.28±0.00	0.46±0.02	0.27±0.00	0.31±0.00	0.58±0.00	0.43±0.01	0.28±0.00	0.28±0.00
Llama3.1-8B	Ans (Base)	Acc (%)	38.50±2.68	73.67±1.55	12.50±1.78	27.22±0.45	54.00±0.71	9.27±0.74	38.47±0.66	61.07±1.76	**77.13±2.04**
		Time(s)	**0.24±0.00**	**0.24±0.00**	**0.25±0.00**	**0.24±0.00**	**0.24±0.00**	**0.25±0.00**	**0.26±0.00**	**0.24±0.00**	**0.24±0.00**
	CoT	Acc (%)	**98.17±0.62**	81.83±1.25	**81.17±2.09**	**98.70±0.52**	**86.50±0.41**	24.93±1.20	**47.13±1.09**	**69.13±2.00**	69.13±0.34
		Time(s)	4.58±0.08	6.37±0.03	4.45±0.03	3.06±0.02	3.03±0.02	8.72±0.04	6.32±0.04	7.45±0.10	6.06±0.04
	CoD	Acc (%)	93.00±0.41	70.67±1.25	58.00±0.82	93.52±0.69	75.17±1.55	**30.53±0.96**	41.87±0.98	67.20±0.71	75.00±0.00
		Time(s)	0.48±0.00	0.46±0.01	1.86±0.03	0.93±0.03	1.25±0.04	5.82±0.17	1.38±0.02	0.84±0.02	0.46±0.01
	TokenSkip	Acc (%)	87.50±4.81	23.67±2.36	31.67±2.72	11.11±3.95	64.83±1.55	10.80±1.85	19.07±1.84	31.00±8.09	46.87±3.03
		Time(s)	1.89±0.04	2.02±0.00	1.93±0.14	2.02±0.01	2.04±0.03	9.34±1.42	2.00±0.04	1.89±0.11	1.98±0.04
	Pause	Acc (%)	57.50±4.26	70.17±3.68	15.00±2.83	81.67±6.35	71.50±2.86	9.67±1.48	36.40±3.05	37.40±15.29	69.47±3.87
		Time(s)	0.15±0.07	0.48±0.17	0.18±0.08	0.13±0.08	0.11±0.05	0.98±0.77	0.45±0.21	0.46±0.18	0.44±0.18
	iCoT-SI	Acc (%)	36.00±7.12	79.50±0.71	15.83±4.11	57.22±7.49	72.67±1.65	11.07±0.74	32.07±4.49	49.00±4.60	73.73±2.00
		Time(s)	0.22±0.09	0.29±0.11	0.22±0.08	0.20±0.08	0.20±0.09	0.97±0.73	0.37±0.12	0.32±0.14	0.37±0.12
	Coconut	Acc (%)	37.33±3.47	64.83±3.09	11.50±3.34	35.00±2.72	56.17±0.62	10.47±1.36	25.87±0.82	16.27±3.70	66.33±0.34
		Time(s)	0.33±0.05	0.52±0.06	0.58±0.09	0.55±0.04	0.41±0.05	3.63±0.88	0.53±0.02	0.50±0.02	0.44±0.04
	CODI	Acc (%)	63.00±6.68	74.50±2.94	8.50±2.16	49.26±8.92	59.00±6.16	7.20±0.28	33.93±0.50	53.87±6.63	68.00±1.07
		Time(s)	0.37±0.00	0.37±0.00	0.37±0.00	0.37±0.00	0.37±0.00	0.38±0.00	0.37±0.00	0.37±0.00	0.37±0.00
	SIM-CoT	Acc (%)	28.67±2.01	60.67±5.39	11.17±1.31	35.00±4.03	55.17±2.72	10.60±0.16	23.93±4.16	24.20±3.74	59.07±5.73
		Time(s)	0.24±0.03	0.52±0.05	0.48±0.04	0.46±0.05	0.37±0.04	3.04±0.74	0.51±0.03	0.48±0.01	0.44±0.05
	SoftCoT	Acc (%)	46.67±5.17	56.00±4.60	9.50±1.08	28.15±3.02	47.50±4.97	8.80±4.97	20.27±4.43	17.60±2.57	53.87±8.88
		Time(s)	0.50±0.07	0.71±0.01	0.72±0.01	0.70±0.01	0.58±0.09	2.92±3.26	0.69±0.01	0.63±0.02	0.66±0.00
	SemCoT	Acc (%)	55.67±15.34	71.50±19.80	11.83±7.19	42.04±29.99	49.17±12.98	19.60±4.42	41.27±23.15	20.47±7.91	70.27±14.97
		Time(s)	0.65±0.16	0.84±0.01	0.87±0.01	0.84±0.01	0.78±0.02	5.41±3.28	0.84±0.01	0.80±0.04	0.74±0.06
	CIRF$_{\text{Full}}$	Acc (%)	77.00±0.82	81.83±0.62	56.17±2.59	**98.52±0.69**	76.50±0.71	11.27±0.09	34.13±0.96	68.93±0.82	74.20±0.71
		Time(s)	0.34±0.01	0.25±0.00	0.57±0.01	0.38±0.00	0.33±0.01	0.76±0.03	0.48±0.01	0.44±0.01	0.33±0.00
	CIRF$_{\text{Fast}}$	Acc (%)	57.17±0.24	81.17±1.43	51.83±1.43	96.11±0.79	**78.17±1.03**	9.33±0.25	35.27±1.36	64.47±0.52	75.73±1.00
		Time(s)	0.16±0.00	0.19±0.00	0.45±0.01	0.29±0.00	0.23±0.00	0.44±0.01	0.30±0.00	0.30±0.01	0.21±0.00
	CIRF$_{\text{Faster}}$	Acc (%)	57.50±0.00	82.67±0.62	45.33±0.85	90.93±0.69	76.17±0.47	9.27±0.38	34.47±0.41	60.53±0.68	75.20±0.49
		Time(s)	0.17±0.00	0.21±0.00	0.39±0.01	0.28±0.00	0.23±0.00	0.32±0.01	0.24±0.00	0.27±0.01	0.20±0.00

Table 8: Accuracy (%) and inference time (seconds) across baselines and CIRF variants (**Full**: $\gamma = 0.0$, **Fast**: $\gamma = 0.1$, **Faster**: $\gamma = 0.2$). Temperature=0.7, mean±std over 3 runs. **Bold** indicates the best result per column within each model.