CRISP: INTERPRETABLE AND ADAPTIVE CHAIN-OF-THOUGHT COMPRESSION VIA SMALL-MODEL COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-Thought (CoT) reasoning substantially improves the performance of large language models on complex tasks, but often produces lengthy and redundant or no-longer-useful intermediate states. However, managing such histories is challenging because reasoning steps differ in semantic utility: some should be preserved verbatim, some can be summarized, some can be removed, and reasoning can occasionally terminate early. Existing approaches delete historical tokens, terminate reasoning early, or compress context into latent representations, but they may require additional training, discard useful information, or produce model-dependent states that are difficult to interpret and reuse. To address these limitations, we introduce CRISP, a step-level framework for semantic reasoning-state management. For each newly generated reasoning step, CRISP selects one of five actions—KEEP, LIGHT, HEAVY, SKIP, or STOP using a lightweight Multi-Agent Controller (MAC) built on a frozen auxiliary language model. The controller separately assesses answer sufficiency, deletion safety, and compression strength, and combines these signals through risk-aware arbitration. For compressible steps, the same auxiliary model rewrites the original reasoning into compact natural-language states that can be re-encoded by the reasoning model, preserving an interpretable reusable, and transferable interface rather than model-specific latent representations. Experiments across multiple benchmarks and model families show that CRISP achieves a favorable accuracy–efficiency trade-off, demonstrating the effectiveness of semantic step-level management for long-chain reasoning.

1 INTRODUCTION

As the parameter scale of Large Language Models (LLMs) continues to expand, these models have demonstrated increasingly strong reasoning capabilities (Zhou et al., 2025; 2026). Chain-of-Thought (CoT) reasoning has become a key mechanism for improving the ability of LLMs to solve complex reasoning tasks, such as mathematical problem-solving, symbolic reasoning, and code generation. Long-chain reasoning continuously accumulates intermediate derivations into the active context: every retained step remains available to subsequent decoding and contributes to KV-cache growth. However, not all intermediate steps remain equally useful. Reasoning models frequently restate established facts, repeat equivalent derivations, perform redundant verification, or retain intermediate explanations whose useful information has already been summarized by later conclusions. As reasoning continues, such states consume context and memory even when their future utility has substantially diminished. This motivates managing not only how much reasoning is generated, but also which previously generated reasoning states should remain active.

Existing approaches improve CoT efficiency by either shortening reasoning trajectories or managing accumulated reasoning histories. Supervised fine-tuning and reinforcement learning encourage shorter reasoning paths but require additional training of the reasoning model (Yu et al., 2025; Xia et al., 2025; Arora & Zanette, 2025; Xiang et al., 2025). Token-budget control directly constrains generation length and may restrict necessary reasoning (Han et al., 2025), while adaptive stopping may terminate before all required intermediate information is obtained (Kang et al., 2025; Liu & Wang, 2025; Li et al., 2026a; Yang et al., 2025a). Alternatively, token-level KV-cache eviction methods, such as H_2O (Zhang et al., 2023) and StreamingLLM (Xiao et al., 2024), may disrupt

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

Table 1: Comparison of efficient reasoning methods. ✓, ✗, and ★denote Yes, No, and Partial, respectively. CRISP compresses previously generated reasoning into an interpretable textual state during decoding, reducing KV-memory growth without retraining the large reasoning model.

Method Category	Stage	KV Mem.	No LLM Train	Interp.	Transfer
CoT length restriction (Xia et al., 2025; Han et al., 2025; Arora & Zanette, 2025)	Decode	★	★	✓	★
Token-level KV eviction (Zhang et al., 2023; Xiao et al., 2024)	Decode	✓	✓	✗	✓
Latent / Gist-token Compression (Zhang et al., 2025; Chen et al., 2025)	Decode	✓	★	✗	★
Speculative decoding (Leviathan et al., 2023; Elhoushi et al., 2024)	Decode	✗	✓	✗	★
Input / Context Compression (Jiang et al., 2023; Chevalier et al., 2023)	Prefill	✗	✓	★	✓
Small-Model Routing (Huang et al., 2026; Zeng et al., 2026)	Decode	★	✓	★	★
Small-Model-Guided Reasoning (Liu et al., 2025; Xu et al., 2025)	Decode	★	★	★	✓
CRISP (Ours)	**Decode**	✓	✓	✓	✓

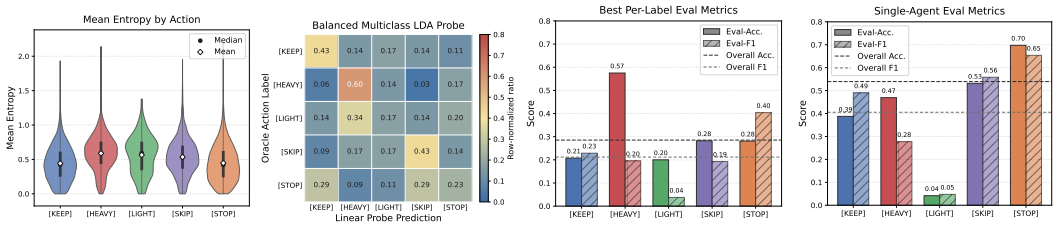
the semantic coherence of reasoning steps, whereas latent or special-token compression produces representations that are difficult to interpret, verify, or reuse across models (Pang et al., 2024; Zhang et al., 2025; Chen et al., 2025). These limitations motivate fine-grained step-level management with interpretable and transferable textual states.

We manage the KV cache at the granularity of reasoning steps. Our key observation is that intermediate steps exhibit heterogeneous semantic *future utility*: some introduce answer-critical variables or constraints, some contain useful but redundant detail, some contribute no unique information, and in some cases the accumulated trajectory is already sufficient for answering. We therefore formulate history management with five executable actions (Section 2)—KEEP, LIGHT, HEAVY, SKIP, and STOP—covering preservation, two levels of compression, deletion, and termination. This formulation poses three challenges. First, step utility is context dependent, so the same content may be dispensable in one trajectory but critical in another. Second, aggressive interventions have asymmetric error costs, requiring conservative decisions under semantic uncertainty. Third, routing, rewriting, and re-encoding introduce non-negligible overhead that must be amortized by reducing the active context. A useful system must therefore jointly balance semantic preservation, memory reduction, and intervention cost.

To address these challenges, we propose CRISP, a semantic step-level reasoning-state manager that operates without updating the reasoning model. CRISP uses a Large Reasoning Model (LRM) for task reasoning and a compact frozen Auxiliary Model (AM) for state assessment and rewriting. Instead of directly predicting the five actions from a single representation, the Multi-Agent Controller (MAC) constructs role-conditioned semantic views for answer sufficiency, deletion safety, redundancy, and compression strength. Lightweight probes (Li et al., 2026b) extract these signals, and a risk-aware Arbiter selects the executable action. When compression is selected, the same AM converts verbose reasoning into discrete natural-language states. These states are then re-encoded by the LRM and replace the corresponding active KV entries. The resulting representation remains readable and model agnostic at the interface level.

Experiments across multiple reasoning benchmarks and model families characterize the trade-offs among answer accuracy, end-to-end latency, and long-horizon memory usage. CRISP substantially reduces active reasoning-state growth, while its end-to-end latency benefit depends on the backbone scale, trajectory length, and intervention frequency. CRISP is also orthogonal to agent-level compression methods for multi-turn or multi-subtask reasoning and can be combined with them to further improve performance.

- We formulate long-chain reasoning history management as a **step-level semantic state-management problem** with five executable actions—KEEP, LIGHT, HEAVY, SKIP, and STOP—covering preservation, compression, deletion, and termination within a unified framework.
- We introduce a **role-factorized and risk-aware controller** built on a frozen auxiliary model, together with an interpretable textual compressor that replaces verbose reasoning with compact natural-language states without modifying the reasoning model.
- We conduct extensive experiments across diverse reasoning benchmarks and model families, demonstrating that CRISP achieves a favorable accuracy–efficiency trade-off and supports cross-model reuse of compressed reasoning states.



(a) (b) (c) (d)

Figure 1: Four routing diagnostics show weak separability across action labels. (a) Entropy statistics show highly overlapping label distributions. (b) Balanced multiclass LDA probing fails to linearly separate action labels. (c) A lightweight MLP router still struggles with fine-grained prediction. (d) A single representation-based judge improves over shallow features but remains insufficient for reliable routing.

2 Background and Motivation

2.1 Step-Level Action Taxonomy

CoT improves complex reasoning by explicitly generating intermediate reasoning steps before producing the final answer (Wei et al., 2022). Formally, given an input problem x, the model generates a reasoning trajectory

$$C = (c_1, \ldots, c_n), c_t \sim p_\theta(c_t \mid x, c_{<t}), y \sim p_\theta(y \mid x, C), \tag{1}$$

where c_t denotes the t-th reasoning step and y is the final answer. We segment each trajectory using consecutive line breaks, i.e., $\backslash\mathrm{n}\backslash\mathrm{n}$, and treat each resulting unit as a reasoning step.

Since reasoning steps differ substantially in their utility, we introduce a unified five-action space covering preservation, compression, deletion, and termination. KEEP preserves necessary information; LIGHT shortens mildly verbose content; HEAVY distills highly redundant but useful information; SKIP removes steps with no unique contribution; and STOP terminates reasoning once the accumulated context is sufficient for answering. Unlike methods specialized for pruning, compression, or early stopping, this formulation jointly manages each step according to its semantic utility and answer sufficiency (Zhang et al., 2025; Yang et al., 2025a).

2.2 Motivation

Selecting the appropriate action is inherently semantic and context-dependent: the same step may require different treatments depending on the input problem, preceding history, and answer-convergence state. To study this challenge, we construct a redundancy-labeled CoT dataset of 3,000 examples and 38,257 reasoning steps covering mathematics, coding, and science. detailed in the Appendix A.

As shown in Figure 1a, lightweight features (entropy-based features) distributions overlap substantially across actions, making threshold-based routing unreliable. A multiclass LDA probe using position, length, entropy, interaction, and history-relative features also exhibits poor separability in Figure 1b. Although an MLP router improves upon these shallow baselines, it still struggles with fine-grained prediction, as shown in Figure 1c.

A large LLM judge could provide stronger semantic assessment, but repeatedly invoking it during generation would incur prohibitive overhead. Following Representation-as-a-Judge (Li et al., 2026b), we further construct a compact single-agent baseline that uses a shared prompt containing all action criteria and predicts the final action with one probe. Its semantic representations outperform handcrafted features but remain insufficient for reliable decisions, as shown in Figure 1d. This suggests that a shared representation may entangle distinct judging criteria and layer preferences, motivating our Multi-Agent Controller with role-specific prompts and specialized probes.

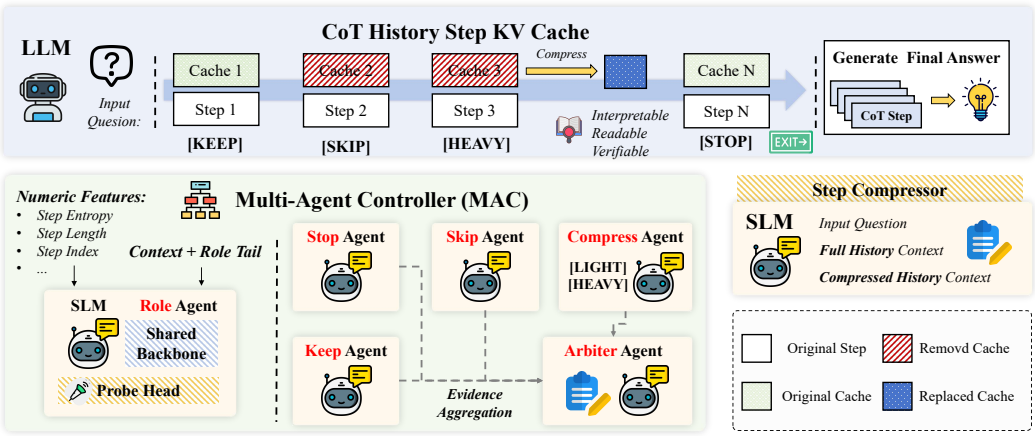


Figure 2: Overview of CRISP. The Large Reasoning Model (LRM) generates CoT steps, while the Auxiliary Model (AM) supports action prediction and compression. After each step, the controller selects KEEP, LIGHT, HEAVY, SKIP, or STOP to update the active reasoning context and its KV states.

3 METHOD

3.1 OVERVIEW

We propose CRISP, a fine-grained framework for managing CoT reasoning during inference. As illustrated in Figure 2, CRISP employs a Large Reasoning Model (LRM) to generate reasoning steps and final answers, together with a compact Auxiliary Model (AM) for action prediction and step compression. After each generated step c_t, the Multi-Agent Controller (MAC) predicts one of five actions: KEEP, LIGHT, HEAVY, SKIP, or STOP. KEEP retains the step, SKIP removes it and its KV states, and STOP terminates reasoning and triggers final-answer generation. For LIGHT and HEAVY, the AM rewrites the step with different compression strengths, and the compressed text is re-encoded by the LRM to replace the original KV states. Through these action-specific updates, CRISP maintains a compact yet informative reasoning context for subsequent generation.

3.2 MULTI-AGENT CONTROLLER

The Multi-Agent Controller (MAC) determines how each newly generated reasoning step should be handled. Since the five executable actions correspond to different semantic questions, we do not formulate action selection as a flat classification problem. Specifically, STOP evaluates whether the accumulated reasoning is sufficient for answering, KEEP, SKIP, and COMPRESS characterize the utility of the current step, and LIGHT and HEAVY further distinguish the compression strength. We therefore factorize the supervision into three dimensions:

$$y_t^S \in \{\text{SOP}, \text{NoSOP}\}, \quad y_t^R \in \{\text{KEEP}, \text{SKIP}, \text{COMPRESS}\}, \quad y_t^G \in \{\text{LIGHT}, \text{HEAVY}\}, \tag{2}$$

where y_t^G is defined only for compressible steps.

This factorization is implemented by four specialist roles: **Stop** evaluates answer sufficiency, **Skip** assesses whether the current step can be safely removed, **Redundancy** determines whether useful information should be preserved or compressed, and **Strength** selects the compression level. Each specialist uses a tailored judge prompt and a role-preferred AM layer to capture its corresponding semantic signal. For the current step c_t, MAC constructs a shared routing context

$$z_t = [x; \widetilde{C}_{t-k:t-1}; c_t], \tag{3}$$

where x is the input question and $\widetilde{C}_{t-k:t-1}$ denotes the recent effective reasoning history. Following Representation-as-a-Judge (Li et al., 2026b), specialist j appends a role-specific tail prompt r_j and predicts its judgment from an intermediate AM representation:

$$h_{t,j}^{(\ell_j)} = \text{AM}_{\ell_j}([z_t; r_j]), \qquad p_{t,j} = \text{softmax}\left(W_j h_{t,j}^{(\ell_j)} + b_j\right). \tag{4}$$

To reduce online overhead, the shared context is encoded once, and all role-specific tails are evaluated in a batch by reusing its prefix KV states.

The Arbiter integrates the specialist predictions and confidence signals to select the final executable action. Because incorrect interventions have asymmetric consequences, MAC follows the semantic-risk ordering

$$\textsc{Stop} \succ \textsc{Skip} \succ \textsc{Heavy} \succ \textsc{Light} \succ \textsc{Keep}, \tag{5}$$

where $\succ$ denotes potential semantic damage rather than computational cost. High-risk actions are selected only when their confidence satisfies action-specific criteria; otherwise, the Arbiter falls back to a more conservative action. The specialist objectives, confidence calibration, and training procedure are detailed in the Appendix B.1.

3.3 Step Compressor

When the MAC selects $a_t \in \{\textsc{Light}, \textsc{Heavy}\}$, CRISP uses the same AM to rewrite the current step c_t into a shorter textual state $\hat{c}_t$. The objective is to remove redundancy while preserving the information required for subsequent reasoning. We maintain the original and compressed reasoning histories:

$$C_t^{\text{ori}} = \left(c_1^{\text{ori}}, \ldots, c_t^{\text{ori}}\right), \qquad C_t^{\text{cmp}} = \left(c_1^{\text{cmp}}, \ldots, c_t^{\text{cmp}}\right). \tag{6}$$

The original history retains complete information for compression, whereas the compressed history constitutes the effective context used by the LRM.

The two compression actions differ in both strength and contextual scope. For Light, the step remains informative but contains mainly surface-level verbosity. The AM therefore uses a local window of recent compressed steps:

$$C_{t,w_l}^{\text{light}} = \left(c_{t-w_l}^{\text{cmp}}, \ldots, c_{t-1}^{\text{cmp}}\right), \tag{7}$$

where w_l denotes the light-context window. This local context supports fidelity-preserving rewriting with limited auxiliary overhead.

For Heavy, the current step is more redundant but may depend on earlier variables, constraints, or intermediate conclusions. We therefore construct a hybrid context:

$$C_{t,w_h}^{\text{hyb}} = \left(c_1^{\text{cmp}}, \ldots, c_{t-w_h-1}^{\text{cmp}}, \quad c_{t-w_h}^{\text{ori}}, \ldots, c_{t-1}^{\text{ori}}\right), \tag{8}$$

where distant history is represented by compressed states and the recent w_h steps are retained in their original form. This preserves the global reasoning state while maintaining detailed local dependencies.

The compressed replacement is generated as:

$$\hat{c}_t = \begin{cases} \text{AM}_{\text{cmp}}\left(x, C_{t,w_l}^{\text{light}}, c_t; \rho_{\textsc{Light}}\right), & a_t = \textsc{Light}, \\ \text{AM}_{\text{cmp}}\left(x, C_{t,w_h}^{\text{hyb}}, c_t; \rho_{\textsc{Heavy}}\right), & a_t = \textsc{Heavy}, \end{cases} \tag{9}$$

where $\rho_{\textsc{Light}}$ and $\rho_{\textsc{Heavy}}$ specify the corresponding compression strengths. Light preserves detailed variables, equations, constraints, and conclusions while removing surface redundancy, whereas Heavy retains only the derived facts and intermediate states needed for future reasoning.

Finally, the LRM re-encodes $\hat{c}_t$ and replaces the original step and its KV states in the active context. This operation reduces the context used by subsequent decoding while maintaining a readable and semantically coherent reasoning trajectory.

3.4 Implementation

The AM remains frozen, and only the lightweight specialist probes are trained. Role-specific tail prompts are distilled from representative positive and negative examples. Each probe is first warmed up with role-specific supervision, after which all specialists and the Arbiter are jointly optimized for five-way action prediction.

Table 2: Accuracy (%) and inference time across five benchmarks and three LRMs. Higher accuracy and lower inference time are better. The best and second-best results among non-Vanilla methods within each LRM are shown in bold and underlined, respectively.

Method	AM	GSM8K		MATH-500		AGIEval		MBPP+		AIME		Avg.	
		Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓	Acc.↑	Time↓
DeepSeek-R1-Distill-Qwen-7B													
Vanilla	–	79.23	4.55	84.20	14.30	26.33	0.53	54.76	2.44	56.67	2.50	60.24	4.86
H_2O	–	0.83	**3.38**	1.40	**1.26**	25.29	0.73	12.96	3.83	3.33	7.02	8.76	3.24
SnapKV	–	4.17	43.27	**85.20**	7.53	26.32	0.52	19.05	**2.17**	3.33	2.32	27.61	11.16
SepLLM	–	74.75	6.39	74.80	21.10	26.08	0.78	40.74	3.37	23.33	3.59	47.94	7.04
DEER	–	77.41	5.97	77.00	6.11	**26.42**	0.60	52.91	2.31	36.67	5.21	54.08	4.04
CRISP	Qwen-0.5B	79.00	3.89	81.60	7.60	26.33	0.52	**54.76**	2.30	30.00	**1.69**	54.34	**3.20**
CRISP	Qwen-1.5B	**79.91**	4.20	82.00	9.67	26.33	**0.51**	**54.76**	2.21	**46.67**	2.39	**57.93**	3.80
DeepSeek-R1-Distill-Llama-8B													
Vanilla	–	78.24	9.68	76.60	13.57	25.85	0.61	50.26	2.56	40.00	3.31	54.19	5.95
H_2O	–	2.88	**3.36**	61.00	34.51	**26.10**	0.83	16.67	4.23	3.33	13.42	22.00	11.27
SnapKV	–	4.17	43.27	73.20	10.45	25.60	**0.58**	1.59	2.55	3.33	**3.12**	21.58	12.00
SepLLM	–	0.61	3.77	1.20	**1.44**	26.00	0.95	0.00	4.32	3.33	14.10	6.23	**4.92**
DEER	–	**76.57**	15.86	77.20	16.75	25.76	0.69	46.56	2.51	**30.00**	5.72	51.22	8.31
CRISP	Qwen-0.5B	75.28	7.42	83.80	11.27	25.85	**0.58**	**50.26**	2.32	**30.00**	3.30	53.04	4.98
CRISP	Qwen-1.5B	75.06	8.35	**85.60**	14.49	25.85	0.59	**50.26**	**2.29**	**30.00**	3.26	**53.35**	5.80
Qwen3-14B-Thinking													
Vanilla	–	79.39	20.74	97.00	20.61	59.37	0.91	86.77	4.01	73.33	4.91	79.17	10.24
H_2O	–	5.38	5.00	49.20	1.65	25.17	1.25	28.04	5.97	3.33	17.53	22.22	6.28
SnapKV	–	3.49	95.27	96.80	20.29	27.41	0.91	12.96	**2.65**	13.33	4.57	30.80	24.74
SepLLM	–	52.92	**4.08**	52.20	**1.60**	58.87	1.40	86.51	5.49	13.33	9.70	52.77	**4.45**
DEER	–	**75.36**	19.89	95.60	19.67	**59.72**	1.20	83.07	4.23	**73.33**	**4.46**	**77.42**	9.89
CRISP	Qwen-0.5B	74.00	20.36	96.20	20.20	59.37	**0.90**	**86.77**	3.06	56.67	4.86	74.60	9.88
CRISP	Qwen-1.5B	74.07	20.27	**97.00**	21.90	59.39	0.91	86.51	2.80	53.33	4.94	74.06	10.16

4 EXPERIMENTS

4.1 EXPERIMENT SETUP

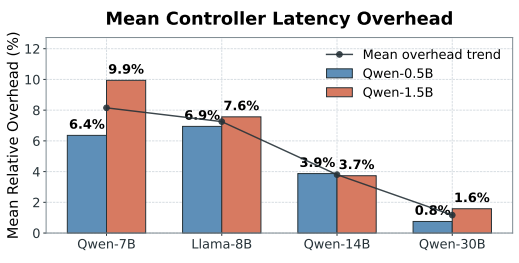
LLMs and Benchmarks. We select DeepSeek-R1 distilled models (DeepSeek-AI, 2025) as the reasoning models, including DeepSeek-R1-Distill-Qwen-7B, DeepSeek-R1-Distill-Llama-8B and QWen3-14B-Thinking. We further employ Qwen2.5-1.5B (Yang et al., 2024) as both the compression model and the semantic extraction model. We evaluate these LLMs on a diverse set of challenging reasoning benchmarks, including GSM8K (Cobbe et al., 2021) under zeroshot CoT prompting, AGIEval (Zhong et al., 2024), MATH500 (Hendrycks et al., 2021) MBPP+ (Austin et al., 2021) instruction-formatted evaluation, and AIME (Jia, 2024). We use lm-evaluation-harness (Gao et al., 2023) to construct open-ended prompts for these benchmarks and report the evaluation results.

Baselines. We compare our method with recent approaches for CoT KV cache management. These baselines include KV-cache eviction methods, such as H_2O (Zhang et al., 2023), SepLLM (Chen et al., 2025), and SnapKV (Li et al., 2024), as well as reasoning early-stopping methods, including DEER (Yang et al., 2025a). Detailed reproduction settings are provided in the Appendix C.1.

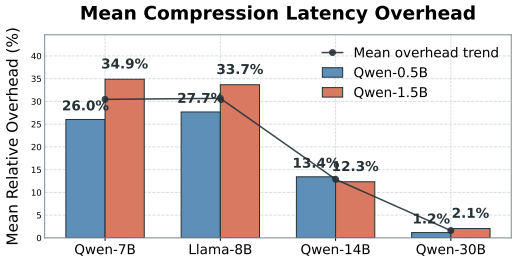
Implementation Details. We retain the MAC with the most recent $k = 8$ historical steps, which is approximately 600 tokens on average, and select different representation layers for different specialist roles; detailed parameter ablations are provided in Appendix C.

4.2 MAIN RESULTS

As shown in Table 2, CRISP generally preserves most of the Vanilla reasoning performance while reducing average inference time across different backbones. It also matches or exceeds Vanilla on several benchmarks, indicating that step-level LIGHT, HEAVY, SKIP, and STOP operations can effectively control KV-cache growth without substantially disrupting reasoning. In contrast, token-level KV eviction baselines suffer severe accuracy degradation on several complex tasks, despite repeated verification. This suggests that directly removing individual KV entries may break semantic dependencies in long reasoning trajectories, whereas CRISP achieves a more stable accuracy–efficiency trade-off through semantically coherent step-level management.



(a) Mean relative latency overhead introduced by controllers.



(b) Mean relative latency overhead introduced by step compressor.

5 Analysis

5.1 Latency Analysis

Controller Overhead. We measure the isolated runtime overhead of MAC under two context lengths and a fixed decoding budget, with the controller invoked every 48 generated tokens. As shown in Figure 3a, MAC introduces only modest overhead, averaging 5.1% and remaining below 10% across all configurations. Its relative cost decreases as the LRM becomes larger, indicating that controller inference is particularly lightweight for larger reasoning models. Detailed results are provided in the Figure 8.

Compression Overhead. We evaluate LIGHT and HEAVY compression under the same decoding setting. As shown in Figure 3b, compression is more expensive than action prediction because it requires an additional autoregressive pass through the AM. Nevertheless, its relative overhead decreases sharply with LRM scale, falling to approximately 1%–2% on Qwen-30B. The 0.5B AM is generally more efficient, whereas the 1.5B AM incurs additional latency in exchange for stronger compression capability. Complete breakdowns are reported in the Figure 9.

5.2 Memory Analysis

We analyze peak GPU-memory growth with a global batch size of 16 and a 32K-token generation budget. As shown in Figure 4, the AM introduces a fixed memory overhead, but this cost is gradually offset as MAC applies LIGHT, HEAVY, and SKIP to reduce the active reasoning context and slow KV-cache growth. At their stopping points, the 0.5B and 1.5B configurations reduce memory by 25.0% and 18.1%, respectively, relative to vanilla inference at the same sequence lengths. The subsequent STOP action further avoids unnecessary decoding. Overall, adaptive step-level management eventually outweighs the AM overhead, with the 0.5B variant benefiting from lower auxiliary-model cost and earlier interventions.

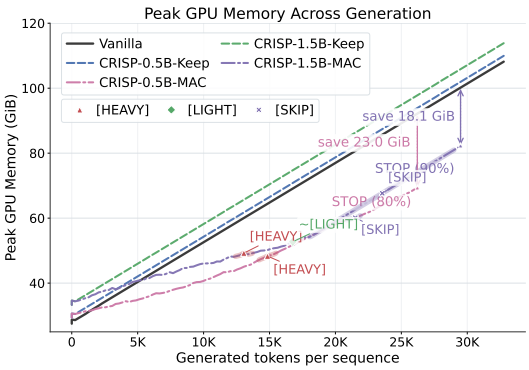


Figure 4: Peak GPU-memory growth during 32K-token generation. MAC progressively reduces active KV-cache memory in generation.

Table 3: Step-level compression quality judged by a 30B model. Ratio is the compressed-to-original token percentage.

AM	Policy	DeepSeek-R1-Distill-Llama-8B					DeepSeek-R1-Distill-Qwen-7B				
		Quality ↑	Faith. ↑	Retain. ↑	Cont. ↑	Ratio (%) ↓	Quality ↑	Faith. ↑	Retain. ↑	Cont. ↑	Ratio (%) ↓
Qwen-0.5B	Alternate	8.47	9.07	8.50	8.59	30.10	8.84	9.52	9.00	9.12	38.73
	HEAVY	8.23	9.04	8.31	8.34	40.16	8.76	9.22	8.82	8.90	31.09
	LIGHT	8.79	9.70	8.87	8.93	33.07	9.01	**9.55**	**9.22**	**9.40**	38.24
Qwen-1.5B	Alternate	9.07	9.44	9.09	9.12	20.35	8.88	9.12	8.90	8.94	18.58
	HEAVY	8.47	8.67	8.45	8.60	20.55	8.82	9.36	8.80	8.88	20.06
	LIGHT	**9.48**	**9.78**	**9.62**	**9.62**	**19.13**	**9.08**	9.39	**9.22**	9.22	**17.85**

Table 4: GSM8K accuracy under forced per-step compression. Δ is measured relative to Vanilla.

AM	Policy	Llama-8B		Qwen-7B	
		Acc. (%) ↑	Δ (pp)	Acc. (%) ↑	Δ (pp)
–	Vanilla	78.24	0.00	79.23	0.00
Qwen-0.5B	LIGHT	76.27	−1.97	76.80	−2.43
	HEAVY	76.27	−1.97	76.88	−2.35
	LIGHT+HEAVY	76.27	−1.97	76.80	−2.43
Qwen-1.5B	LIGHT	**77.18**	−1.90	**75.59**	−2.05
	HEAVY	76.19	−2.05	76.88	−2.35
	LIGHT+HEAVY	76.27	−1.97	77.11	−2.12

Table 5: Cross-model transfer of reasoning states generated by Qwen-7B to other models.

Method	Target: Llama-8B			Target: Qwen14B		
	AGIEval	MATH500	Avg.	AGIEval	MATH500	Avg.
Self-CoT	70.0	70.0	70.0	85.0	90.0	87.5
Recent-256	**75.0**	80.0	77.5	60.0	85.0	72.5
Attention-256	15.0	40.0	27.5	25.0	45.0	35.0
H2O-256	65.0	80.0	72.5	60.0	85.0	72.5
Compress-0.5B	65.0	75.0	70.0	65.0	80.0	72.5
Compress-1.5B	65.0	**85.0**	75.0	65.0	**90.0**	**77.5**
CRISP-0.5B	70.0	**85.0**	77.5	65.0	85.0	75.0
CRISP-1.5B	**75.0**	**85.0**	**80.0**	**70.0**	80.0	75.0

5.3 CASE STUDY

As shown in Figure 5, CRISP removes four repeated setup steps and compresses two verbose reasoning steps, while retaining the critical numerical derivation after validator rejection. When it obtains the final answer, it correctly terminates the thinking and generates the final answer. Consequently, the committed trajectory is reduced from 2,147 to 1,554 tokens (27.6%) without affecting the final answer. This case illustrates how CRISP balances reasoning compression and correctness preservation. More cases are shown in Appendix C.6.

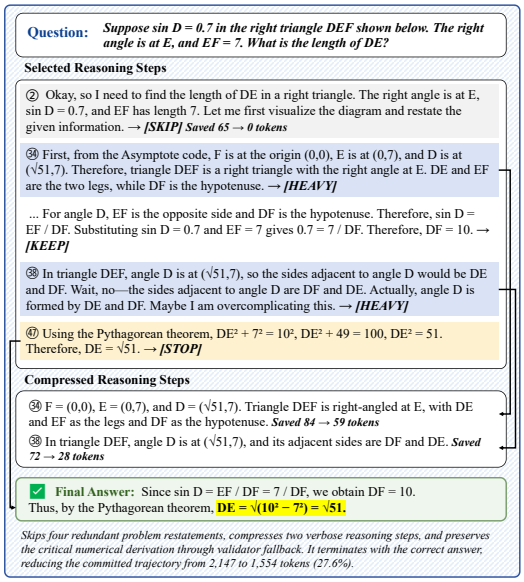


Figure 5: A successful CRISP trajectory. Four redundant setup steps are skipped, two verbose steps are compressed, and early stop when thinking is sufficient.

5.4 COMPRESSION ANALYSIS

Step-level Compression Quality We evaluate whether the AM preserves the semantics of individual reasoning steps using a 30B LLM judge, which scores overall quality, factual faithfulness, information retention, and reasoning continuity. As shown in Table 3, according to the 30B judge, the compressed steps receive consistently high local faithfulness and retention scores while substantially reducing token length. These results provide model-based evidence that the compressor often preserves information judged important at the individual-step level. The 1.5B AM generally achieves a better compression–quality trade-off, whereas LIGHT produces the strongest local quality. The less consistent performance of HEAVY further supports applying aggressive compression only to highly redundant steps.

End-to-end Accuracy under Forced Compression. We further force every intermediate step on GSM8K to follow a fixed compression policy. As shown in Table 4, all configurations remain within approximately 2.5 percentage points of Vanilla accuracy. This result shows that the compressed textual states preserve most information required for subsequent reasoning, even under continuous per-step compression.

5.5 TRANSFERABILITY ANALYSIS

We evaluate whether textual reasoning states compressed from Qwen-7B can be reused by Llama-8B and Qwen-14B without transferring hidden states or KV caches. As shown in Table 5, CRISP-1.5B achieves the best transferred-history performance on Llama-8B, with 80.0% overall accuracy. Its advantage over token-level eviction methods highlights the importance of preserving coherent step-level semantics for cross-model reuse. On Qwen-14B, Self-CoT remains stronger overall, likely due to the target model's greater capability. Nevertheless, Compress-1.5B matches Self-CoT on MATH-500 and performs best among transferred-history methods overall. These results show that natural-

language compressed states are reusable across model architectures and scales, although stronger target models may favor more conservative transfer.

6 Ablation Study

We further analyze the sensitivity of history window of MAC and role-specific representation layers, with complete results provided in Appendix C.5. Across history windows of 5–10 steps, controller performance remains relatively stable. We therefore use $k = 8$ as a balanced choice between contextual coverage and routing overhead. Layer ablations further reveal clear role-dependent differences: STOP and SKIP exhibit the strongest semantic separability, redundancy prediction is moderately separable, whereas compression-strength prediction remains more challenging.

7 Related Work

7.1 Efficient Chain-of-Thought Reasoning

Existing methods improve CoT efficiency either by shortening reasoning trajectories or by reducing the cost of accumulated reasoning histories. The former includes supervised fine-tuning (Yu et al., 2025; Xia et al., 2025), token-budget control (Han et al., 2025), reinforcement learning with length-aware objectives (Arora & Zanette, 2025; Xiang et al., 2025), and adaptive early stopping (Kang et al., 2025; Liu & Wang, 2025; Li et al., 2026a; Yang et al., 2025a). These approaches reduce generated reasoning tokens, but may require additional training, constrain the original reasoning behavior, or terminate reasoning prematurely.

The latter direction manages historical states during decoding (Li et al., 2024). Token-level eviction methods, such as H_2O (Zhang et al., 2023) and StreamingLLM (Xiao et al., 2024), discard less important KV entries, while AnLLM (Pang et al., 2024), LightThinker (Zhang et al., 2025), and SepLLM (Chen et al., 2025) compress context into anchor, gist, or separator-token representations. In contrast, CRISP manages reasoning at the step level and rewrites redundant history into discrete natural-language states that remain interpretable and reusable, without retraining the Large Reasoning Model.

7.2 Small-Large Model Collaboration for Efficient Reasoning

Small and large models have also been combined to improve inference efficiency. Speculative decoding methods use a small model (Leviathan et al., 2023) or shallow layers (Elhoushi et al., 2024) to draft tokens for large-model verification. Context-compression methods, including LLMLingua (Jiang et al., 2023), AutoCompressors (Chevalier et al., 2023), RECOMP (Xu et al., 2023), and EXIT (Hwang et al., 2025), reduce prompts, retrieved evidence, or long inputs via a small model before generation. Other approaches employ small models for KV selection (Zhao et al., 2025), attention prediction (Yang et al., 2025b), reasoning guidance (Liu et al., 2025), soft reasoning (Xu et al., 2025), or routing (Huang et al., 2026; Zeng et al., 2026).

Unlike these methods, which primarily use the smaller model to draft, select, guide, or route future generations, CRISP uses a compact Auxiliary Model both to evaluate previously generated reasoning steps and to rewrite compressible steps into interpretable textual states.

8 Conclusion

We presented CRISP, a step-level framework for managing CoT reasoning through five actions: KEEP, LIGHT, HEAVY, SKIP, and STOP. A lightweight Multi-Agent Controller uses role-specific representations from a shared Auxiliary Model to select actions, while the same model generates interpretable textual compressions without retraining the Large Reasoning Model. Experiments across multiple benchmarks and model families show that CRISP provides an effective accuracy–efficiency trade-off through semantic reasoning-context management.

Limitations and Future Work. CRISP reduces only subsequent decoding cost and does not recover the computation used to generate the original step or optimize input prefill. Future work will extend the framework to prefill optimization and multi-turn agent memory management.

9 AI USE STATEMENT

In this work, generative AI tools were used to assist with language polishing and literature search and review, including improving the readability of the manuscript, identifying potentially relevant literature, and summarizing background material. They were not used to generate experimental data, formulate mathematical claims or proofs, design the proposed methodology or experiments, implement the core method, process experimental data, or interpret the reported experimental results. All AI-suggested references and factual statements were independently checked against the original sources, and all AI-assisted text was reviewed and revised by the authors. We take full responsibility for the final content, claims, and results presented in this work.

10 ETHICS STATEMENT

This work evaluates reasoning-efficiency methods using publicly available language models and reasoning benchmarks, and constructs step-level supervision from model-generated reasoning traces. It does not involve human subjects or the collection of private, personal, or sensitive information. Because CRISP may remove or rewrite intermediate reasoning content, incorrect compression or routing decisions can potentially alter downstream model behavior. The proposed framework should therefore not be regarded as a safety mechanism and should be carefully validated before use in safety-critical or high-stakes applications. CRISP also inherits the safety, bias, and misuse risks of the underlying language models and automated model-based evaluators.

11 REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the CRISP framework and experimental protocol to support reproducibility. Sections 2.2 and 3 define the step-level action taxonomy, Multi-Agent Controller, Step Compressor, context construction, and inference procedure, while Sections 3.4 and 4.1 describe the controller setup, model backbones, benchmarks, baselines, and main implementation settings. The Appendix further documents dataset curation and labeling, controller training, baseline reproduction, prompts, parameter studies, and additional experimental details. Section 5 reports the protocols used for latency, compression-quality, memory, and cross-model transfer analyses. All publicly available models and benchmark datasets used in the experiments are cited in the paper.

REFERENCES

Daman Arora and Andrea Zanette. Training language models to reason efficiently, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. SepLLM: Accelerate large language models by compressing one segment into one separator. In *Forty-second International Conference on Machine Learning*, 2025.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

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. 2021.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. 2025.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early

exit inference and self-speculative decoding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 12622–12642, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics*, pp. 24842–24855, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Chengsong Huang, Tong Zheng, Langlin Huang, Jinyuan Li, Haolin Liu, and Jiaxin Huang. Relayllm: Efficient reasoning via collaborative decoding, 2026.

Taeho Hwang, Sukmin Cho, Soyeong Jeong, Hoyun Song, SeungYoon Han, and Jong C Park. Exit: Context-aware extractive compression for enhancing retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4895–4924, 2025.

Maxwell Jia. Aime problem set 2024, 2024. URL https://huggingface.co/datasets/Maxwell-Jia/AIME_2024.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, 2023.

Liwei Kang, Yue Deng, Yao Xiao, Zhanfeng Mo, Wee Sun Lee, and Lidong Bing. First try matters: Revisiting the role of reflection in reasoning models, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the International Conference on Machine Learning*, volume 202, pp. 19274–19286, 2023.

Gengyang Li, Wang Cai, Yifeng Gao, and Yunfang Wu. Syncthink: A training-free strategy to align inference termination with reasoning saturation, 2026a.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Proceedings of Advances in Neural Information Processing Systems*, 2024.

Zhuochun Li, Yong Zhang, Ming Li, Yuelyu Ji, Yiming Zeng, Ning Cheng, Yun Zhu, Yanmeng Wang, Shaojun Wang, Jing Xiao, and Daqing He. Rethinking llm-as-a-judge: Representation-as-a-judge with small language models via semantic capacity asymmetry, 2026b.

Xin Liu and Lu Wang. Answer convergence as a signal for early stopping in reasoning, 2025.

Yule Liu, Jingyi Zheng, Zhen Sun, Zifan Peng, Wenhan Dong, Zeyang Sha, Shiwen Cui, Weiqiang Wang, and Xinlei He. Thought manipulation: External thought can be efficient for large reasoning models, 2025.

Jianhui Pang, Fanghua Ye, Derek Wong, Xin He, Wanshun Chen, and Longyue Wang. Anchor-based large language models. In *Proceedings of Findings of the Association for Computational Linguistics*, pp. 4958–4976, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms, 2025.

Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. Recomp: Improving retrieval-augmented lms with compression and selective augmentation, 2023. URL https://arxiv.org/abs/2310.04408.

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. 2024.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models, 2025a.

Qingyue Yang, Jie Wang, Xing Li, Zhihai Wang, Chen Chen, Lei Chen, Xianzhi Yu, Wulong Liu, Jianye HAO, Mingxuan Yuan, and Bin Li. Attentionpredictor: Temporal patterns matter for KV cache compression. In *Proceedings of Annual Conference on Neural Information Processing Systems*, 2025b.

Bin Yu, Hang Yuan, Haotian Li, Xueyin Xu, Yuliang Wei, Bailing Wang, Weizhen Qi, and Kai Chen. Long-short chain-of-thought mixture supervised fine-tuning eliciting efficient reasoning in large language models, 2025.

Wenhao Zeng, Xuteng Zhang, Yuling Shi, Chao Hu, Yuting Chen, Beijun Shen, and Xiaodong Gu. Glimprouter: Efficient collaborative inference by glimpsing one token of thoughts, 2026.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. Lightthinker: Thinking step-by-step compression, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023.

Yi Zhao, Yajuan Peng, Nguyen Cam-Tu, Zuchao Li, Wang Xiaoliang, hai zhao, and Xiaoming Fu. SmallKV: Small model assisted compensation of KV cache compression for efficient LLM inference. In *Proceedings of Annual Conference on Neural Information Processing Systems*, 2025.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 2024.

Yuhua Zhou, Ruifeng Li, Changhai Zhou, Fei Yang, and Aimin PAN. BSLoRA: Enhancing the parameter efficiency of loRA with intra-layer and inter-layer sharing. In *Proceedings of International Conference on Machine Learning*, 2025.

Yuhua Zhou, Changhai Zhou, Shiyang Zhang, Fei Yang, Yi Zhang, and Aimin Pan. Lara: Layer-wise rank allocation for efficient fine-tuning of pruned large language models. *Information Processing & Management*, 63(3):104538, 2026.

APPENDIX

A DATASET CURATION

To analyze step-level redundancy and stopping opportunities in long-chain reasoning, we construct a motivation dataset with 3,000 examples. We sample source problems from Bespoke-Stratos-17k [1], a reasoning dataset containing questions, reasoning traces, and answers, and perform stratified sampling over three task types: mathematics, coding, and science. For each selected example, we extract the system prompt, question, reference solution, and reference answer. We then prompt a reasoning model M_r (here, we use DeepSeek-R1-Distill-Qwen-7B) to generate a chain-of-thought trajectory using greedy decoding with a maximum reasoning budget of 1,024 tokens.

The generated trajectory is segmented using consecutive line breaks, i.e., `\n\n`, following the step definition used throughout the paper. For each step, we record its token length and token-level predictive entropy,

$$H_t = -\sum_{v \in \mathcal{V}} p_t(v) \log p_t(v),$$

and aggregate the entropy within each step using mean, maximum, minimum, and median values. We further annotate each step with two types of supervision. First, a judge model M_j (here, we use Qwen3-30B-Instruct) assigns a redundancy label from $\{\texttt{KEEP}, \texttt{LIGHT}, \texttt{HEAVY}, \texttt{SKIP}\}$, indicating whether the step should be preserved, lightly compressed, heavily compressed, or removed. Second, to identify whether reasoning can stop at a given step, we append a closing thinking token to the current prefix, let M_r generate the final answer, and ask M_j to judge whether it matches the reference answer. If the answer is correct, the step is labeled as a valid stopping point (`SOP`); otherwise, it is labeled `NOSOP`.

Table 6: Statistics of the annotated CoT dataset.

Task Type	Examples	Ratio	Steps	Avg. Steps
Math	1,200	40.00%	10,813	9.01
Code	1,000	33.33%	19,973	19.97
Science	800	26.67%	7,471	9.34
Total	3,000	100.00%	38,257	12.75

Overall, from Table 6 and Table 7, the dataset contains 38,257 reasoning steps, with an average of 12.75 steps per example. The average generated reasoning length is 712.56 tokens per example, and each step contains 54.91 tokens on average. Among all examples, 2,213 final answers are judged correct, corresponding to a final-answer accuracy of 73.77%. Moreover, 2,762 examples, or 92.07%, contain at least one valid stopping point, showing that many reasoning trajectories can be terminated earlier without changing the final answer.

Table 7: Step-level label statistics. Avg. Tokens denotes the average number of generated tokens in each reasoning step, and Avg. Entropy denotes the mean token predictive entropy within a step.

Category	Label	Count	Ratio	Avg. Tokens	Avg. Entropy
Redundancy	KEEP	25,732	67.26%	54.36	0.422
Redundancy	HEAVY	6,199	16.20%	69.78	0.591
Redundancy	LIGHT	618	1.62%	42.13	0.539
Redundancy	SKIP	5,708	14.92%	42.66	0.493
Stopping	SOP	25,743	67.29%	53.84	0.447
Stopping	NOSOP	12,514	32.71%	57.12	0.494
All	–	38,257	100.00%	54.91	0.462

[1] https://huggingface.co/datasets/bespokelabs/Bespoke-Stratos-17k

B IMPLEMENTATION

B.1 MULTI-AGENT CONTROLLER

Factorized label formulation. For each reasoning step c_t, the controller takes

$$z_t = [x; \widetilde{C}_{t-k:t-1}; c_t; m_t], \tag{10}$$

where x is the input question, $\widetilde{C}_{t-k:t-1}$ is the recent effective reasoning history, and m_t denotes optional numerical features. The supervision is decomposed into three dimensions:

$$\begin{aligned} y_t^S &\in \{\text{SOP}, \text{NoSOP}\}, \\ y_t^R &\in \{\text{KEEP}, \text{SKIP}, \text{COMPRESS}\}, \\ y_t^G &\in \{\text{LIGHT}, \text{HEAVY}\}, \end{aligned} \tag{11}$$

where y_t^G is defined only when $y_t^R = \text{COMPRESS}$. Answer sufficiency and step utility are independently annotated; the hierarchy is introduced only when selecting the final executable action.

Specialist prediction and arbitration. MAC contains STOP, SKIP, REDUNDANCY, and STRENGTH specialists. Each role appends a tail prompt r_j to the shared context and extracts a representation from layer ℓ_j of the frozen Auxiliary Model:

$$h_{t,j}^{(\ell_j)} = \text{AM}_{\ell_j}([z_t; r_j]), \qquad p_{t,j} = \text{softmax}\left(W_j h_{t,j}^{(\ell_j)} + b_j\right). \tag{12}$$

The common prefix is encoded once, while the role tails are evaluated in a batch using the shared prefix KV cache. We denote the probabilities of answer sufficiency, safe deletion, compression, and light compression as p_t^S, p_t^D, p_t^C, and p_t^L, respectively.

These probabilities define a structured base distribution:

$$\begin{aligned} P_{\text{base}}(\text{STOP}) &= p_t^S, \\ P_{\text{base}}(\text{SKIP}) &= (1-p_t^S)p_t^D, \\ P_{\text{base}}(\text{KEEP}) &= (1-p_t^S)(1-p_t^D)(1-p_t^C), \\ P_{\text{base}}(\text{HEAVY}) &= (1-p_t^S)(1-p_t^D)p_t^C(1-p_t^L), \\ P_{\text{base}}(\text{LIGHT}) &= (1-p_t^S)(1-p_t^D)p_t^C p_t^L. \end{aligned} \tag{13}$$

The Arbiter further combines the specialist representations, probabilities, and confidence features to predict the final five-way action distribution $\pi_t = P_\phi(a_t \mid z_t)$.

Risk-aware inference. The final action follows the intervention-risk ordering

$$\text{STOP} \succ \text{SKIP} \succ \text{HEAVY} \succ \text{LIGHT} \succ \text{KEEP}. \tag{14}$$

Using calibrated specialist or Arbiter probabilities, inference is performed as

$$\hat{a}_t = \begin{cases} \text{STOP}, & p_t^S \geq \tau_S, \\ \text{SKIP}, & p_t^S < \tau_S \wedge p_t^D \geq \tau_D, \\ \text{KEEP}, & p_t^S < \tau_S \wedge p_t^D < \tau_D \wedge p_t^C < \tau_C, \\ \text{HEAVY}, & p_t^C \geq \tau_C \wedge (1-p_t^L) \geq \tau_H, \\ \text{LIGHT}, & \text{otherwise}. \end{cases} \tag{15}$$

The thresholds for STOP, SKIP, and HEAVY are calibrated for high precision on the validation set, while τ_C controls the overall compression–preservation trade-off.

KV Management. Let $\widetilde{C}_{t-1}$ and K_{t-1} denote the active reasoning context and its KV cache before generating step c_t. After action selection, CRISP updates the active state as

$$(\widetilde{C}_t, K_t) = \begin{cases} ([\widetilde{C}_{t-1}; c_t],\ K_t^{\text{ori}}), & a_t = \text{KEEP}, \\ ([\widetilde{C}_{t-1}; \hat{c}_t],\ K_{t-1} \oplus \kappa(\hat{c}_t \mid \widetilde{C}_{t-1})), & a_t \in \{\text{LIGHT}, \text{HEAVY}\}, \\ (\widetilde{C}_{t-1},\ K_{t-1}), & a_t = \text{SKIP}, \end{cases} \tag{16}$$

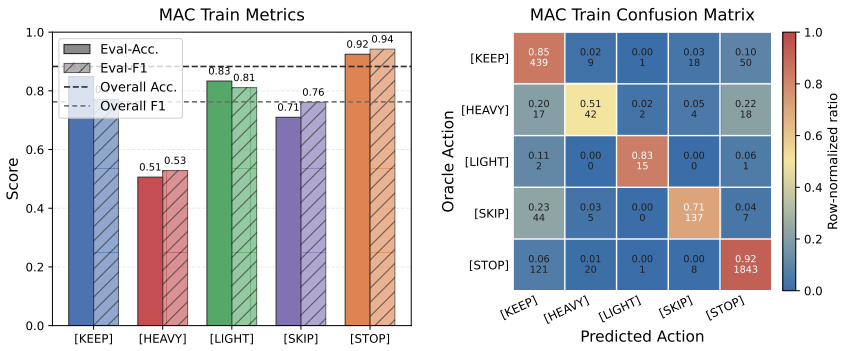


Figure 6: Per-label training metrics and confusion matrix of the MAC.

where K_t^{ori} is the cache produced while generating c_t, $\kappa(\cdot)$ denotes LRM re-encoding, and $\oplus$ denotes cache concatenation. Thus, SKIP rolls back the current-step KV states, while LIGHT/HEAVY reuse the prefix cache and re-encode only the compressed replacement. For STOP, no further CoT step is generated and the model proceeds to final-answer generation.

Two-stage training. The AM backbone remains frozen throughout training. In Stage I, we optimize the role-specific probes using factorized semantic supervision:

$$\mathcal{L}_{\text{spec}} = \lambda_S \mathcal{L}_S + \lambda_D \mathcal{L}_D + \lambda_R \mathcal{L}_R + \lambda_G m_t^C \mathcal{L}_G, \tag{17}$$

where $\mathcal{L}_S$ predicts answer sufficiency, $\mathcal{L}_D$ predicts deletion safety, $\mathcal{L}_R$ predicts $\{\text{KEEP}, \text{SKIP}, \text{COMPRESS}\}$, and $\mathcal{L}_G$ distinguishes LIGHT from HEAVY. The mask

$$m_t^C = \mathbb{I}[y_t^R = \text{COMPRESS}] \tag{18}$$

ensures that the strength loss is applied only to compressible steps.

In Stage II, the Arbiter aggregates specialist representations, probabilities, and confidence features. Its objective is

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{spec}} + \lambda_A \mathcal{L}_{\text{action}} + \lambda_{\text{sub}} \mathcal{L}_{\text{arb}} + \lambda_{\text{con}} \mathcal{L}_{\text{cons}}, \tag{19}$$

where $\mathcal{L}_{\text{action}}$ supervises the final five-way action, $\mathcal{L}_{\text{arb}}$ supervises the Arbiter's factorized semantic heads, and $\mathcal{L}_{\text{cons}}$ encourages agreement between specialist and Arbiter predictions. Action-specific thresholds are calibrated on the validation set, with higher precision requirements for STOP, SKIP, and HEAVY.

training performance. We evaluate the fitted MAC on the training split of the annotated CoT dataset. As shown in Figure 6, MAC achieves an overall accuracy of 0.88 and a macro-F1 score of 0.76, and eval accuracy of 0.77. STOP, KEEP, and LIGHT obtain recalls of 0.92, 0.85, and 0.83, respectively, whereas SKIP reaches 0.71 and HEAVY remains the most difficult action at 0.51. The confusion matrix shows that most residual errors involve HEAVY and SKIP. Some compressible or removable steps are conservatively retained as KEEP.

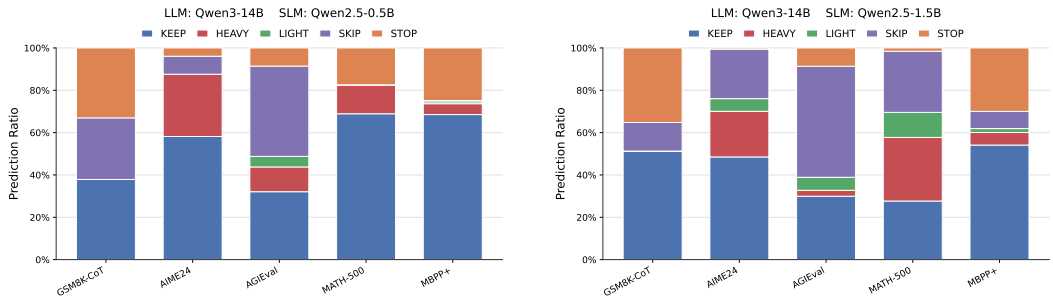
C EXPERIMENTS

C.1 BASELINE REPRODUCTION

We compare CRISP with four efficiency-oriented baselines: H_2O, SepLLM, DEER, and SnapKV. All baselines are evaluated under the same backbone models, datasets, prompts, decoding settings, maximum reasoning budget, and answer-extraction scripts as CRISP whenever applicable. We implement all methods as inference-time interventions without retraining or modifying the Large Reasoning Model. For KV-cache or context-compression baselines, we align the effective cache or token budget with our setting; for early-stopping baselines, we keep the same maximum generation budget and final-answer evaluation protocol.

- **H_2O** is a KV-cache eviction method that retains recent tokens and heavy-hitter tokens selected by accumulated attention scores, reducing memory usage during long-form generation. We reproduce H_2O based on its open-source repository,[2] and integrate its dynamic KV-cache eviction policy into our inference pipeline.
- **SepLLM** accelerates inference by compressing token segments into separator tokens, allowing long contexts to be represented by initial tokens, separator tokens, and recent tokens. We reproduce SepLLM based on its open-source repository,[3] and adapt its separator-cache mechanism to our CoT reasoning benchmarks.
- **DEER** is a dynamic early-exit method for reasoning models that periodically checks whether the accumulated reasoning is sufficient to produce a confident final answer. We reproduce DEER based on its open-source repository,[4] and evaluate its confidence-based stopping strategy under the same prompting, decoding, and answer-extraction protocol as CRISP.
- **SnapKV** is a training-free KV-cache compression method that selects important prompt tokens according to attention patterns observed before generation, reducing subsequent decoding cost. We reproduce SnapKV based on its open-source repository,[5] and plug its cache-selection module into the same inference and benchmark pipeline.

C.2 Controller Action Distribution Analysis



(a) Action distribution across tasks of 0.5B AM backbone.

(b) Action distribution across tasks of 1.5B AM backbone.

We analyze controller actions across five benchmarks using Qwen3-14B as the LRM and Qwen2.5-0.5B or Qwen2.5-1.5B as the AM. As shown in Figure 7a and Figure 7b, the distributions vary substantially across tasks. GSM8K-CoT contains relatively short trajectories and therefore triggers more STOP decisions, whereas AGIEval receives a higher proportion of SKIP decisions, suggesting greater step-level redundancy under the learned policy. In contrast, the low STOP ratio on AIME24 reflects its longer and more difficult derivations, while MATH-500 receives more compression interventions. The 1.5B AM generally selects fewer KEEP actions and more LIGHT, HEAVY, and SKIP actions than the 0.5B AM, indicating a more aggressive management policy. Overall, the controller adapts its decisions to both task characteristics and AM capability.

C.3 Latency Analysis

Controller Overhead. Figure 8 provides the complete latency breakdown for all base models, controller sizes, and context lengths. The isolated benchmark uses context lengths of 512 and 1,024 tokens and requires each base model to generate 2,048 tokens. The controller is invoked every 48 generated tokens, resulting in 42 controller invocations per decoding run. We compare the total wall-clock latency of controller-enabled decoding with that of a baseline run without the controller. Each reported result is averaged over three independent runs.

[2] https://github.com/FMInference/H2O
[3] https://github.com/HKUDS/SepLLM
[4] https://github.com/iie-ycx/DEER
[5] https://github.com/FasterDecoding/SnapKV

For Qwen-7B, the 0.5B controller introduces an overhead of 6.33% and 6.38% at context lengths 512 and 1,024, respectively. The corresponding overheads for the 1.5B controller are 9.90% and 9.99%. On Llama-8B, the overhead ranges from 6.70% to 7.18% with the 0.5B controller and from 7.25% to 7.87% with the 1.5B controller. The relative overhead becomes substantially smaller for larger base models. On Qwen-14B, the 0.5B and 1.5B controllers introduce overheads of 3.08%–4.66% and 3.25%–4.21%, respectively. On Qwen-30B, the corresponding overheads further decrease to 0.59%–0.92% and 1.54%–1.62%. Across all configurations, a single invocation takes approximately 49–52 ms for the 0.5B controller and 58–62 ms for the 1.5B controller.

Overall, the controller introduces a small and bounded runtime cost. The relative overhead consistently remains below 10% and decreases with the size of the base model, indicating that controller inference is particularly inexpensive relative to long-form decoding with larger models.

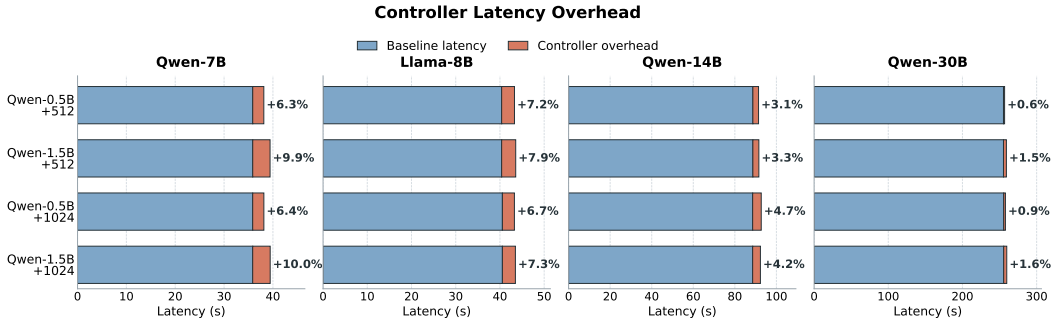


Figure 8: Complete controller latency breakdown across base models, controller sizes, and context lengths. The blue portion represents the baseline decoding latency, while the orange portion represents the additional end-to-end latency measured with the controller enabled. Each result is averaged over three independent runs.

Compression Overhead. Figure 9 provides the complete latency breakdown for all LRMs, AM sizes, compression modes, and context lengths. The benchmark uses context lengths of 512 and 1,024 tokens, keeps eight reasoning steps in the context, and requires each base model to generate 2,048 tokens. Compression is invoked every 144 generated tokens, resulting in 14 compression invocations per decoding run. We compare the total wall-clock latency of compression-enabled decoding with that of a baseline run without compression. The LIGHT and HEAVY modes are evaluated separately, and each reported result is averaged over three independent runs.

For Qwen-7B, the 0.5B summary model introduces an overhead ranging from 22.51% to 29.36% across the two context lengths and compression modes. The corresponding overhead for the 1.5B summary model ranges from 34.48% to 35.24%. On Llama-8B, the overhead ranges from 24.85% to 31.15% with the 0.5B summary model and from 33.31% to 33.86% with the 1.5B summary model. For Qwen-14B, the corresponding overheads decrease to 12.07%–14.94% and 10.49%–14.43%, respectively. The relative overhead becomes substantially smaller for the largest base model. On Qwen-30B, the 0.5B and 1.5B summary models introduce overheads of 0.10%–2.12% and 0.70%–3.50%, respectively.

Averaged across summary-model sizes, compression modes, and context lengths, the mean overhead is 30.5% for Qwen-7B, 30.7% for Llama-8B, 12.9% for Qwen-14B, and 1.6% for Qwen-30B. The average latency of a single compression invocation ranges from approximately 641–863 ms for the 0.5B summary model and 837–960 ms for the 1.5B summary model.

Overall, compression introduces a substantially larger runtime cost than controller inference because each compression step requires an additional autoregressive generation pass by the summary model. Nevertheless, the relative overhead decreases sharply as the base model becomes larger, since the compression cost becomes smaller relative to the base-model decoding time. The results also show that the choice of summary-model size and compression mode affects the overhead, with the 1.5B summary model generally incurring a higher cost than the 0.5B model.

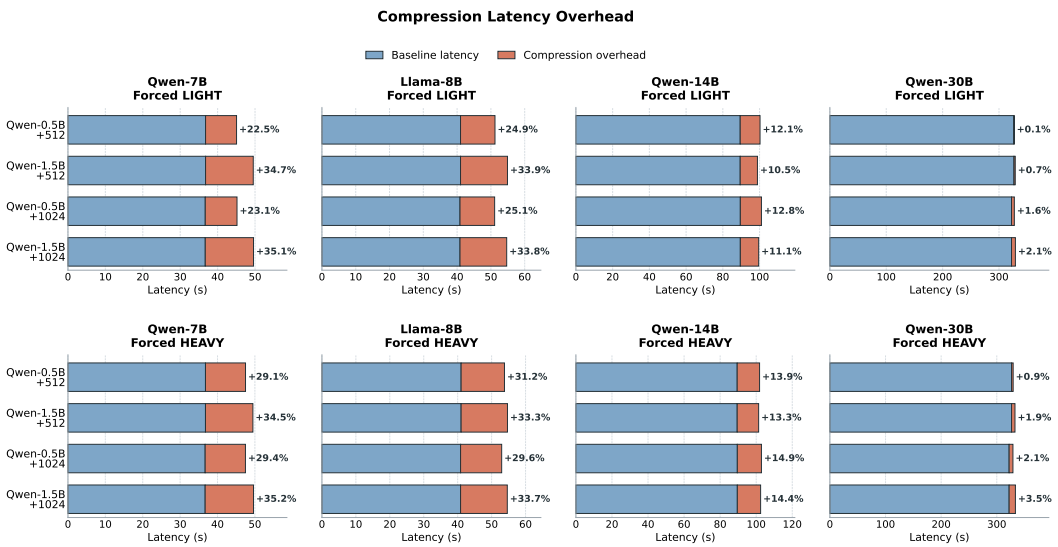


Figure 9: Complete compression latency breakdown across base models, summary-model sizes, compression modes, and context lengths. The blue portion represents the baseline decoding latency, while the orange portion represents the additional end-to-end latency introduced by compression. Each result is averaged over three independent runs.

Table 8: Ablation on the controller history-window length (*history steps*).

History Steps	Qwen-0.5B		Qwen-1.5B	
	Policy F1	Acc.	Policy F1	Acc.
5	66.02	71.67	65.52	**72.02**
6	68.41	75.79	62.91	67.40
7	67.10	74.26	65.39	71.57
8	66.67	73.80	65.20	69.74
9	**69.27**	75.94	64.59	69.33
10	68.75	**76.25**	**65.94**	71.57

C.4 Effect of the History Window of MAC

We conduct an ablation study to investigate how the history-window size affects the decision quality of the MAC. The results are reported in Table 8. Overall, MAC shows relatively stable performance across different window sizes, indicating that its policy decisions are not overly sensitive to this hyperparameter. Meanwhile, slightly longer histories tend to provide better results, suggesting that additional recent reasoning steps can offer useful context for controller decisions. For Qwen-0.5B, the best Policy F1 is achieved with a history window of 9, while the best accuracy is obtained with a window of 10. For Qwen-1.5B, the best Policy F1 is achieved with a history window of 10.

C.5 Layer Ablation of the CRISP Controller

We conduct ablation study on the probe layers used by each role-specific component of the CRISP controller. For the binary source probes (STOP, SKIP, and STRENGTH), we select checkpoints by balanced accuracy. For the multi-class redundancy probe and the five-way arbiter, we use macro-F1. Negative layer indices denote layers counted from the top of the backbone.

Table 9 reports all tested source-layer configurations. The STOP and SKIP probes provide the clearest signals, with best balanced accuracies of 0.7204 and 0.7096, respectively. Redundancy prediction is moderate, with a best macro-F1 of 0.5458. The STRENGTH probe is the weakest source component: although $\{-10, -12\}$ obtains the highest balanced accuracy (0.5580), it has very low recall (0.1558), so we use $\{-8, -12\}$ for its better recall/F1 trade-off.

Table 9: Source probe layer ablation. For binary roles, the score is balanced accuracy; for redundancy, the score is macro-F1. † marks the selected checkpoint used by the arbiter-stage experiment.

Role	Layers	Score	Acc.	F1
STOP	$\{-1\}$	**0.7204**	0.7347	0.5567
STOP	$\{-1,-2\}^{\dagger}$	0.7175	0.7498	**0.5573**
STOP	$\{-1,-4\}$	0.7175	0.7498	**0.5573**
STOP	$\{-4\}$	0.7124	0.7129	0.5437
STOP	$\{-8\}$	0.7060	0.7438	0.5430
STOP	$\{-4,-8,-12\}$	0.6996	0.7340	0.5337
STOP	$\{-8,-12,-16\}$	0.6944	**0.7506**	0.5305
STOP	$\{-1,-2,-4\}$	0.6884	0.7219	0.5189
STOP	$\{-12\}$	0.6856	0.6835	0.5116
STOP	$\{-16\}$	0.6817	0.7069	0.5095
SKIP	$\{-8,-12\}^{\dagger}$	**0.7096**	0.7732	**0.5814**
SKIP	$\{-8\}$	0.7078	0.7754	0.5791
SKIP	$\{-4\}$	0.7069	0.7777	0.5780
SKIP	$\{-1,-4,-8\}$	0.6983	0.7581	0.5645
SKIP	$\{-4,-8\}$	0.6969	0.7596	0.5624
SKIP	$\{-4,-8,-12\}$	0.6969	**0.7898**	0.5634
SKIP	$\{-1\}$	0.6944	0.7476	0.5586
SKIP	$\{-8,-12,-16\}$	0.6920	0.7754	0.5552
REDUNDANCY	$\{-4,-8\}^{\dagger}$	**0.5458**	**0.5536**	**0.5458**
REDUNDANCY	$\{-2,-4,-8\}$	0.5406	0.5486	0.5406
REDUNDANCY	$\{-2,-4\}$	0.5353	0.5437	0.5353
REDUNDANCY	$\{-6,-8\}$	0.5352	0.5397	0.5352
REDUNDANCY	$\{-1,-8,-12\}$	0.5287	0.5327	0.5287
REDUNDANCY	$\{-4,-8,-10\}$	0.5274	0.5327	0.5274
REDUNDANCY	$\{-4,-6,-8\}$	0.5251	0.5367	0.5251
REDUNDANCY	$\{-4,-8,-12\}$	0.5244	0.5327	0.5244
REDUNDANCY	$\{-4,-6\}$	0.5203	0.5397	0.5203
REDUNDANCY	$\{-8\}$	0.5145	0.5288	0.5145
REDUNDANCY	$\{-4\}$	0.5054	0.5288	0.5054
REDUNDANCY	$\{-12\}$	0.4939	0.5060	0.4939
REDUNDANCY	$\{-8,-12,-16\}$	0.4571	0.5020	0.4571
REDUNDANCY	$\{-12,-16,-20\}$	0.4336	0.4474	0.4336
STRENGTH	$\{-10,-12\}$	**0.5580**	**0.7558**	0.2449
STRENGTH	$\{-8,-12\}^{\dagger}$	0.5461	0.6040	**0.3548**
STRENGTH	$\{-8,-10,-12\}$	0.5417	0.6931	0.2791
STRENGTH	$\{-6,-8,-10,-12\}$	0.5377	0.7063	0.2521
STRENGTH	$\{-1,-8,-12\}$	0.5148	0.7360	0.1111
STRENGTH	$\{-8,-12,-16\}$	0.5148	0.7360	0.1111
STRENGTH	$\{-8,-10\}$	0.5128	0.6436	0.2603
STRENGTH	$\{-6,-8\}$	0.5093	0.5809	0.3060
STRENGTH	$\{-6,-8,-10\}$	0.5065	0.7492	0.0256
STRENGTH	$\{-12\}$	0.5039	0.7261	0.0879
STRENGTH	$\{-8\}$	0.4976	0.7360	0.0244
STRENGTH	$\{-4,-8,-12\}$	0.4956	0.7393	0.0000
STRENGTH	$\{-4\}$	0.4925	0.6964	0.1154

Table 10: Representative arbiter layer ablation. Macro-F1 is computed over STOP, SKIP, HEAVY, LIGHT, and KEEP.

Layers	Macro-F1	Acc.	STOP F1	SKIP F1	HEAVY F1	LIGHT F1	KEEP F1
$\{-4\}$	**0.3581**	**0.4318**	0.4687	**0.4621**	**0.3387**	0.0211	0.5000
$\{-12\}$	0.3356	0.4145	0.4270	0.4032	0.3273	0.0000	0.5207
$\{-8\}$	0.3301	0.4062	0.4584	0.3901	0.3006	0.0000	0.5011
$\{-1\}$	0.3240	0.4002	0.3724	0.3738	0.3283	0.0238	0.5216
$\{-1,-8,-12\}$	0.2657	0.3557	**0.5095**	0.0266	0.2767	0.0190	0.4968
$\{-1,-4,-8,-12\}$	0.1677	0.2826	0.1108	0.0000	0.2732	0.0182	0.4365

Table 10 reports the arbiter ablation. The best arbiter uses a single layer, $\{-4\}$, reaching 0.3581 action macro-F1. Other single-layer settings are close but consistently lower, while multi-layer settings degrade substantially. This suggests that the final action decision benefits from a compact layer representation rather than combining multiple heterogeneous depths. The remaining weakness is recall for the high-risk STOP/SKIP actions and the near-failure of the LIGHT class.

C.6 Case Study Analysis

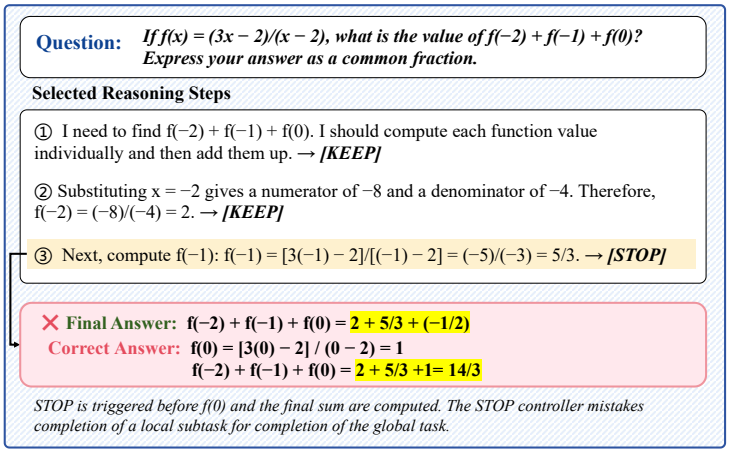


Figure 10: Premature STOP after local subtask completion. The controller terminates after computing f(-1), despite the unresolved evaluation of f(0) and the missing final summation.

Table 11: Role-specific judging criteria used by the Multi-Agent Controller.

Role	Judging criterion
KEEP Agent	Judges whether the current step contains necessary information that should be preserved without modification.
SKIP Agent	Judges whether the current step contributes little or no new information and can be removed from the active context.
STOP Agent	Judges whether the accumulated reasoning context is already sufficient to produce the final answer.
COMPRESS Agent	Judges whether the current step is useful but verbose, and determines whether light or heavy compression is needed.

Failure Case: Premature STOP. Figure 10 illustrates a representative failure caused by an incorrect STOP decision. The model first constructs a complete plan to evaluate $f(-2)$, $f(-1)$, and $f(0)$ before summing the three values. However, after correctly obtaining $f(-2) = 2$ and $f(-1) = 5/3$, the controller issues STOP, even though $f(0)$ and the final summation remain unresolved. This suggests that the controller may confuse completion of a local subtask with completion of the global problem. Since an erroneous STOP decision is irreversible, future designs should incorporate problem-level progress tracking or a final-answer-aware stopping guard.

D ROLE TAILS AND COMPRESSION PROMPTS

This section presents the role-specific tail prompts used by the Multi-Agent Controller (MAC) and the compression prompts used by the Step Compressor. For each routing decision, the shared routing context consists of the input question, recent reasoning history, and the current step to be judged. A role-specific tail is appended to this shared context to induce a corresponding judge-like representation from the frozen AM. The final decision is made by lightweight probes over the extracted representations rather than by directly decoding textual judgments.

STOP Agent Role Tail

Independent STOP Skill. Decide only whether the accumulated state can now answer the original question correctly.
Do not judge whether the current step is verbose, repeated, useful, compressible, or removable.
Use `[SOP]` only when the final answer or a complete extraction path is present; use `[NOSOP]` when any computation, verification, branch, unit, option, code behavior, or factual qualifier is still missing.

STOP requires the answer itself or a direct extraction path with all cases, units, option mapping, and verification already resolved.
Concrete examples:

- Question: "Mia has 12 apples and buys 5 more. How many now?" History: "Start with 12." Current step: "12+5=17, so Mia has 17 apples." Judgment: `[SOP]`.
- Question: "What is the rectangle area?" History: "Area = length $\times$ width." Current step: "The length is 8." Width is still missing. Judgment: `[NOSOP]`.
- Question: "Which option is correct?" History: "A and C remain possible." Current step: "C violates the parity condition, so A is correct." Judgment: `[SOP]`.
- Question: "Return the bug fix." History: "The queue can become empty." Current step: "We should handle that case." The exact implementation is missing. Judgment: `[NOSOP]`.

Valid labels: `[SOP]`, `[NOSOP]`.

SKIP Agent Role Tail

Independent SKIP Skill. Decide only whether deleting the current step loses meaningful information.
Do not decide final answer readiness and do not choose compression strength.
Use `[SKIP]` only when the step is an exact duplicate, filler, meta-commentary, prompt restatement, off-track text, or an unperformed check.
Use `[NOSKIP]` if it adds, repairs, verifies, or disambiguates any value, equation, constraint, relation, branch, option, unit, identifier, or final extraction.
A step can be removable even if the problem is not solved, but removability must be judged only by unique signal in this step.
Concrete examples:

- History already states "$x = 7$." Current step: "So yes, x is 7, as said before." No new signal. Judgment: `[SKIP]`.
- History has "$x + y = 12$" and "$y = 5$." Current step: "Substitute $y = 5$, giving $x = 7$." New substitution. Judgment: `[NOSKIP]`.
- Current step: "I should double-check the arithmetic." No check is performed. Judgment: `[SKIP]`.
- Current step: "Checking: $18 + 24 = 42$, matching the total." Verification is performed. Judgment: `[NOSKIP]`.

Valid labels: `[SKIP]`, `[NOSKIP]`.

REDUNDANCY Agent Role Tail

Independent Redundancy-Route Skill. Decide how the current step should be preserved if reasoning continues.
`[KEEP]` means preserve verbatim because the step carries unique answer-critical signal. `[COMPRESS]` means preserve the useful signal in shorter form. `[SKIP]` means delete because no unique useful signal remains.
Do not let STOP sufficiency force `[SKIP]`; a final-answer step can still be `[KEEP]`.
Final-answer text, unit conversion, and option mapping are usually `[KEEP]` unless duplicated exactly.
Concrete examples:

- Current step: "Since $y = 5$, $x = 12 - 5 = 7$." This is the only explicit substitution. Judgment: `[KEEP]`.
- Current step: "Sorting first is useful because duplicates become adjacent, making a scan possible..." Useful but wordy algorithmic idea. Judgment: `[COMPRESS]`.
- Current step: "We need solve the problem carefully." No task-specific reasoning. Judgment: `[SKIP]`.

- Current step: "Return -1 when the queue empties before finding the target." Behavior-changing code detail. Judgment: `[KEEP]`.

Valid labels: `[KEEP]`, `[COMPRESS]`, `[SKIP]`.

STRENGTH Agent Role Tail

Independent Compression-Strength Skill. This role applies only to useful steps that are already considered compressible.
`[LIGHT]` keeps important details in compact form. `[HEAVY]` keeps only the gist.
Do not use this role as deletion, and do not use `[HEAVY]` for first-time exact facts.
Do not compress away final values, units, or option letters. Use `[LIGHT]` if those details can remain compactly.
Concrete examples:

- Current step: "n must be even and $n \geq 6$ because smaller even values fail." Exact inequality matters. Judgment: `[LIGHT]`.
- Current step: "This approach works because it repeatedly narrows possibilities in the same way described above." Repeated gist is enough. Judgment: `[HEAVY]`.
- Current step: "Use `visited.add(node)` before pushing neighbors into queue." Identifier and order matter. Judgment: `[LIGHT]`.
- Current step repeats the already-kept sorting intuition in several sentences with no new bound or identifier. Judgment: `[HEAVY]`.

Valid labels: `[HEAVY]`, `[LIGHT]`.

ARBITER Agent Role Tail

Final Arbiter Skill. Integrate specialist evidence into exactly one action.
Action meanings:

- `[STOP]` ends reasoning because the accumulated state can answer the original question.
- `[SKIP]` deletes the current step because it has no unique useful signal.
- `[KEEP]` preserves the current step verbatim because it carries answer-critical or uniquely useful signal.
- `[LIGHT]` compresses while preserving important details such as numbers, entities, inequalities, units, code identifiers, branch order, option labels, or boundary cases.
- `[HEAVY]` compresses aggressively when only a repeated or obvious gist is needed.

Decision order: first test STOP sufficiency; if not stopping, test whether deletion is harmless; if deletion is unsafe, choose KEEP for first-time exact signal, LIGHT for detail-bearing compressible signal, and HEAVY for repeated useful gist.
When STOP/SKIP/HEAVY could erase a correction, equation, branch, option choice, final value, or factual qualifier, choose a preserving action.
Avoid premature `[STOP]`; if answer readiness is partial, route the step with `[KEEP]`, `[LIGHT]`, `[HEAVY]`, or `[SKIP]` instead.
Concrete examples:

- Question asks final count. Current step: "$12+5 = 17$, so the answer is 17." All evidence is complete. Decision: `[STOP]`.
- Current step: "I will now solve it step by step." It adds no unique reasoning. Decision: `[SKIP]`.
- Current step: "Substitute $y = 5$ into $x + y = 12$, so $x = 7$." First exact derivation. Decision: `[KEEP]`.
- Current step: "The valid case is n even with $n \geq 6$; the long explanation repeats the parity argument." Details matter but text can shrink. Decision: `[LIGHT]`.
- Current step repeats the already-kept sorting intuition in several sentences with no new identifier or bound. Decision: `[HEAVY]`.

- Current step: "Actually the sign should be negative, so the final value is -3." Correction and final value are unique. Decision: `[KEEP]` unless STOP is the configured final action for complete answer readiness.

Valid labels: `[STOP]`, `[SKIP]`, `[HEAVY]`, `[LIGHT]`, `[KEEP]`.

Light Compression Prompt

You are a concise reasoning-step compressor.
Given the input question, the recent compressed reasoning history, and the current reasoning step, rewrite the current step into a shorter natural-language form.
Requirements:

- Preserve all useful intermediate results, equations, constraints, variable definitions, units, option labels, code identifiers, branch conditions, and conclusions.
- Remove mild verbosity, repeated explanations, redundant verification, and unnecessary wording.
- Keep the compressed step faithful to the original reasoning.
- Do not introduce new facts, new assumptions, or new reasoning results.
- Do not change numerical values, signs, units, entities, option mappings, or code behavior.
- Output only the compressed reasoning step.

Compression strength: `LIGHT`.

Heavy Compression Prompt

You are an aggressive reasoning-step compressor.
Given the input question, a hybrid reasoning context containing distant compressed steps and recent original steps, and the current reasoning step, rewrite the current step into a highly compact natural-language form.
Requirements:

- Preserve only the information necessary for future reasoning or final-answer generation.
- Keep essential intermediate results, equations, constraints, final values, units, option labels, code identifiers, branch conditions, and conclusions.
- Remove redundant derivations, repeated statements, failed attempts, unnecessary verification, and non- essential explanatory prose.
- Make the compressed step substantially shorter than the original step.
- Do not introduce new facts, new assumptions, or new reasoning results.
- Do not alter numerical values, signs, units, entities, option mappings, or code behavior.
- Output only the compressed reasoning step.

Compression strength: `HEAVY`.