ConCISE: Confidence-guided Compression in Step-by-step Efficient Reasoning

Anonymous ACL submission

Abstract

Large Reasoning Models (LRMs) perform strongly in complex reasoning tasks via Chain-of-Thought (CoT) prompting, but often suffer from verbose outputs, increasing computational overhead. Existing fine-tuning-based compression methods either operate post-hoc pruning, risking disruption to reasoning coherence, or rely on sampling-based selection, which fails to remove redundant content thoroughly. To address these limitations, this work begins by framing two key patterns of redundant reflection in LRMs—*Confidence Deficit*, wherein the model reflects on correct intermediate steps, and *Termination Delay*, where reflection continues after a verified, confident answer—through a confidence-guided perspective. Based on this, we introduce CONCISE (**Con**fidence-guided **C**ompression **I**n **S**tep-by-step **E**fficient Reasoning), a framework designed to generate concise reasoning chains, integrating *Confidence Injection* to boost reasoning confidence, and *Early Stopping* to terminate reasoning when confidence is sufficient. Extensive experiments demonstrate that compared to baseline methods, fine-tuning LRMs on CONCISE-generated data yields a better balance between compression and task performance, reducing length by up to ~50% under SimPO, while maintaining high task accuracy.

1 Introduction

In recent years, Large Language Models (LLMs) have achieved remarkable progress in natural language processing, particularly in complex reasoning tasks. Large Reasoning Models (LRMs), such as OpenAI-o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025), which leverage the Chain-of-Thought paradigm (Wei et al., 2022), have demonstrated state-of-the-art performance on benchmarks involving mathematical problem solving and logical reasoning(Muennighoff et al., 2025; Qwen Team, 2024). However, a prominent limitation of
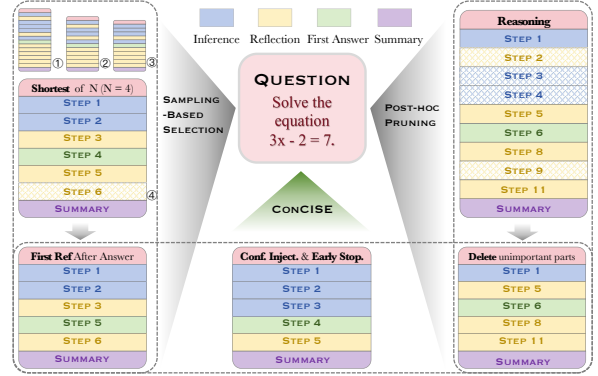

Figure 1: Training dataset construction workflows: CONCISE (our proposed method) vs. existing methods.
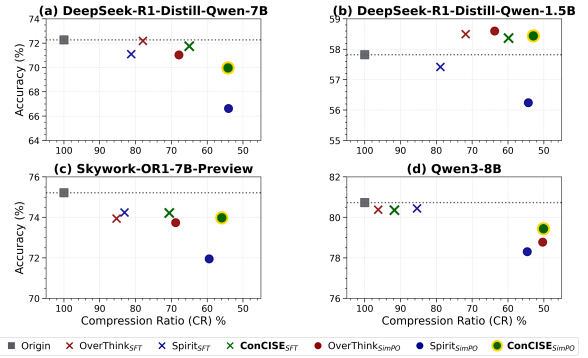

Figure 2: CONCISE achieves a better trade-off between compression and task performance than baselines.

LRMs is their tendency to generate excessively verbose reasoning chains (Feng et al., 2025; Chen et al., 2024), incuring considerable computational overhead, and posing challenges for deployment in resource-constrained settings (Team et al., 2025).

To mitigate LRM output verbosity, recent research focuses on compressing their reasoning chains (Qu et al., 2025; Sui et al., 2025). A prominent strategy involves fine-tuning LRMs on concise reasoning datasets, enabling them to generate shorter responses (Ma et al., 2025; Chen et al., 2024; Munkhbat et al., 2025). The effectiveness of this compression strategy hinges on the training

dataset design. As shown in Figure 1, common sampling-based selection, which generates multiple candidates and picks the shortest correct one, or further removes post-answer redundant reflections (Team et al., 2025; Chen et al., 2024), lacks control during generation, potentially leaving unnecessary steps and reducing compression effectiveness. Another approach, post-hoc pruning, identifies and removes redundant or less important steps from reasoning chains (Cui et al., 2025; Xia et al., 2025), risks disrupting reasoning coherence and degrading the performance of LRMs after fine-tuning.

To overcome existing limitations, we aim to construct compact, coherent reasoning chains as training datasets by precisely removing redundant reflections, ensuring LRMs do not suffer performance degradation after fine-tuning. To this end, based on the understanding that reflections are not solely determined by correctness (Yang et al., 2025), we thus adopt a **confidence-guided** perspective to understand the generation of reflection steps in LRM's reasoning processes. This perspective offers a clear interpretation of two key patterns of redundancy: *Confidence Deficit*, where low internal confidence causes models to undertrust and reflect on correct intermediate steps; and *Termination Delay*, where reflection persists despite a repeatedly verified answer. These patterns inflate reasoning chains and provide actionable insights for how to create concise reasoning chains.

Therefore, we propose CONCISE (**Con**fidence-guided **C**ompression **I**n **S**tep-by-step **E**fficient reasoning), a framework that leverages a confidence-guided perspective for constructing concise reasoning data by actively suppressing redundant reflection during generation. CONCISE features two complementary components designed to mitigate *Confidence Deficit* and *Termination Delay*, respectively: *Confidence Injection*, which inserts tailored phrases before potential reflection points to boost the model's internal confidence, consequently curbing unnecessary reflection; and *Early Stopping*, which employs a lightweight confidence detector to monitor internal confidence after an answer is reached, terminating reasoning upon achieving sufficient confidence. The synergy of these mechanisms enables CONCISE to produce more efficient and compact reasoning chains.

We then fine-tune four mainstream LRMs using CONCISE-generated data via SFT (Wei et al., 2021) and SimPO (Meng et al., 2024), evaluating their performance across multiple reasoning bench-marks. As shown in Figure 2, Experimental results demonstrate that CONCISE consistently achieves a superior trade-off between reasoning compression and task performance under both SFT and SimPO settings compared to baselines. Notably, CONCISE reduces average response length by approximately 50% under SimPO while maintaining high accuracy. Our subsequent analysis reveals that CONCISE's unique training data design enables LRMs, after fine-tuning, to learn to strategically avoid generating redundant reflection steps without harming essential critical reasoning content, thereby achieving efficient reasoning compression while task performance is well maintained.

2 Related Work

Recent research has increasingly focused on mitigating verbosity and redundancy in reasoning chains generated by LRMs, which often produces long outputs that increase computational costs, and even degrade accuracy (Wu et al., 2025; Nayab et al., 2024; Wang et al., 2025). To address this problem, existing approaches can be broadly categorized into three classes: Input-based, Decoding-based, and Model-based (Sui et al., 2025).

Input-based Methods aim to promote concise reasoning by modifying input texts (Lee et al., 2025). Common techniques include imposing token limits in prompts or instructing the model to reason briefly (Han et al., 2024; Renze and Guven, 2024). However, these approaches often fail when the model does not consistently follow these instructions. Another approach, task routing, dynamically selects an appropriate model or reasoning strategy based on input characteristics like question complexity (Ong et al., 2024; Chuang et al., 2025; Aytes et al., 2025). Its effectiveness, however, depends on the accurate assessment of input characteristics, which is not always reliable.

Decoding-based Methods intervene during decoding to control the reasoning process. One common approach compresses steps into latent representations rather than explicit text, which improves brevity but sacrifices interpretability (Hao et al., 2024; Shen et al., 2025b). Another strategy uses dynamic decoding, evaluating each reasoning step and deciding whether to modify or discard it during generation (Sun et al., 2024; Zhang et al., 2025; Xu et al., 2025). While effective at reducing verbosity, these methods introduce additional computational overhead and may disrupt the reasoning coherence.
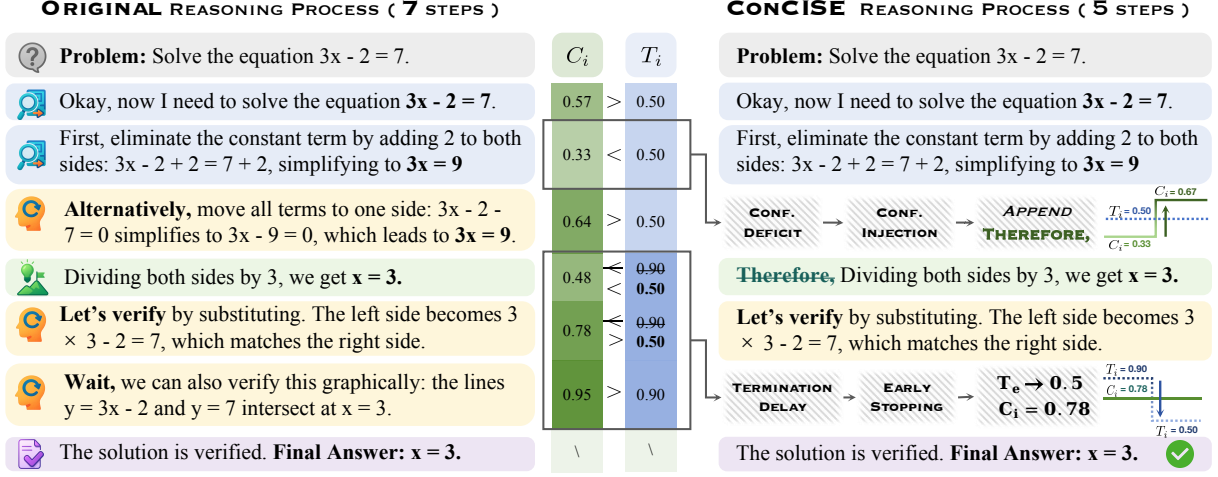
Figure 3: Illustration of CONCISE's confidence-guided approach: identifying patterns (*Confidence Deficit*, *Termination Delay*) and applying mechanisms (*Confidence Injection*, *Early Stopping*) to suppress redundant reflections, shown in contrast to the original reasoning process. C_i denotes step confidence and T_i its threshold.

Model-based Methods train models to generate concise reasoning directly. One approach employs reinforcement learning with reward functions that penalize verbosity (Shen et al., 2025a; Aggarwal and Welleck, 2025; Luo et al., 2025; Arora and Zanette, 2025; Yu et al., 2025), but can be sensitive to reward formulation and impose significant costs. Another common approach is to fine-tune LRMs on datasets with concise reasoning chains(Ma et al., 2025; Cui et al., 2025; Xia et al., 2025). These datasets are often created by selecting the shortest correct chain or removing redundant parts post-hoc. Such methods may inadvertently retain redundant reflection steps or remove useful context, degrading compression efficiency or model performance.

Motivated by the limitations of existing methods, we propose CONCISE, a model-based method that precisely identifies and actively suppresses redundant reflection steps throughout the reasoning process. This approach enables the model to generate efficient reasoning chains while preserving reasoning coherence. Leveraging CONCISE, we construct a high-quality training dataset and fine-tune the model to adopt this concise reasoning ability without compromising model performance.

3 Method

3.1 Confidence-guided Formulation

Reflections in LRMs are not solely triggered by correctness; in many cases, reflection steps are invoked even on correct steps that have been verified (Yang et al., 2025). This suggests that reflection behavior is also linked to the model's internal confidence about current reasoning rather than correctness alone. To explain this, we adopt a **confidence-guided** perspective to formalize when and why LRMs engage in reflection.

Let $S_i = \{s_1, s_2, \ldots, s_i\}$ denote the partial reasoning chain up to step i, where each s_i is a textual reasoning unit. We associate each step s_i with a confidence score $c_i \in [0, 1]$, representing the model's internal confidence of that step. The generation policy of the LRM, denoted by π_θ, maps the current reasoning context S_i to the next step s_{i+1}.

To model the decision between proceeding and reflecting, we introduce a dynamic threshold $t_i \in [0, 1]$, which may vary with the model or the context. At each step, the model generates a `ReflectionStep` as s_{i+1} if its current internal confidence c_i falls below the threshold t_i (i.e., $c_i < t_i$). From this perspective, we further analyze the reflection behavior of LRMs and formally articulate two key patterns responsible for reflection-related redundancy existing in the reasoning process: *Confidence Deficit* and *Termination Delay*.

3.2 Two Key Patterns of Redundancy

Confidence Deficit. One major source of redundancy in LRMs stems from their tendency to under-trust their correct intermediate steps. LRMs often display unexpected reflection despite exhibiting fine-grained reasoning capabilities and achieving high stepwise accuracy, triggering reflection even on simple and unambiguous reasoning steps. This self-undermining behavior leads to redundant reflection with minimal semantic gain. We refer to this phenomenon as *Confidence Deficit*. Formally,

3

Confidence Deficit can be defined as the occurrence of a step s_i satisfying:

$$\exists i, \quad c_i < t_i \quad \text{despite } s_i \text{ being correct.}$$

Termination Delay. LRMs exhibit another important redundant reasoning pattern we term *Termination Delay*: After producing a confident final answer, the model is expected to conclude with minimal additional reasoning. However, it often continues to generate unnecessary reflection steps even after repeatedly verifying the same conclusion. This behavior can be attributed to a sharp increase in the confidence threshold t_i after reaching the answer at step i', making it increasingly difficult for the model's internal confidence c_i to exceed t_i. As a result, even when c_i becomes relatively high, it may still fall short of the heightened t_i, leading to verbose post-answer reasoning. Formally, *Termination Delay* can be characterized as:

$$\exists i > i', \ \forall j < i', \quad t_i \gg t_j \text{ and } c_i < t_i$$

3.3 The CONCISE Framework

To mitigate the above patterns—*Confidence Deficit* and *Termination Delay*—we propose CONCISE, a framework that dynamically steers the reasoning generation process to reduce redundant reflections and produce concise reasoning chains. CONCISE integrates two mechanisms: *Confidence Injection*, which actively inserts confidence phrases to suppress unnecessary reflection steps, and *Early Stopping*, which halts generation once sufficient post-answer confidence is detected. These two mechanisms operate together to generate concise reasoning chains without compromising inference quality. The overall process is illustrated in Figure 3.

3.3.1 Confidence Injection

To alleviate *Confidence Deficit*, we design a simple yet effective mechanism called *Confidence Injection*. The key idea is to boost the model's internal confidence during reasoning by inserting designed phrases, referred to as *confidence phrases*, to prevent unnecessary reflection steps. Considering that indiscriminate injection could disrupt reasoning, we selectively insert confidence phrases at critical points where the model's confidence is relatively low, that is, when $c_i < t_i$, a condition that signifies an impending reflection step.

Specifically, at each step s_i, the model first generates $s_{i+1} = \pi_\theta(S_i)$. If s_{i+1} is a reflection step,
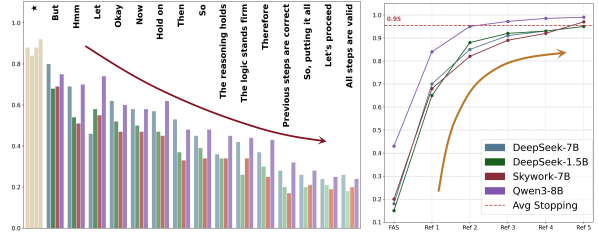
(a) Reflection probability of next step after phrase injection. (b) Confidence calculated by our detector.

Figure 4: Effectiveness and necessity of Confidence Injection and Termination Delay(details in Appendix B.1).

we retroactively modify the input S_i by appending a confidence phrase p_i sampled from a curated pool $\mathcal{P}$, and regenerate s_{i+1}. Formally, the updated generation process is defined as:

$$s_{i+1} = \pi_\theta(S_i) \quad \text{if } c_i \geq t_i \text{ else} \quad \pi_\theta(S_i, p_i)$$

We manually constructed the initial version of $\mathcal{P}$. To evaluate and further refine the initial pool, we conducted experiments measuring the effectiveness of different phrases. We inserted each candidate phrase before the reflection steps and recorded the probability that the model still generated a reflection step. This probability indicates whether the updated confidence c_i^* (after phrase insertion) exceeds the dynamic threshold t_i. As shown in Figure 4a, different phrases exhibit varying reflection rates. Notably, even the best-performing phrases still yield a reflection rate around 20%, suggesting that confidence injection reduces redundant reflections while still retaining necessary verifications. We then selected 20 phrases with the lowest reflection rates to form the final confidence phrase pool. The composition of the phrase pool and detection of reflection steps are provided in Appendix A.1.

3.3.2 Early Stopping

Although Confidence Injection effectively boosts the model's confidence during reasoning, it does not fully mitigate *Termination Delay*, which is caused by the high threshold t_i after the model reaches an answer, leading the model to continue unnecessary reflections even when its confidence is already high. To address this, we design an *Early Stopping* mechanism based on direct estimation of the model's internal confidence.

Specifically, we construct a lightweight confidence detector to provide a quantitative proxy for the model's internal confidence about the answer. We introduce a probing prompt after the current

4

reasoning context and analyze the probability distribution of generated continuations. Based on the statistical analysis shown in Appendix A.2, we collect a set of confidence-indicative phrases $\mathcal{W}_+$, which reflect affirmations of high certainty. The detected confidence score $\hat{c}_i$ at step i is calculated as:

$$\hat{c}_i = \sum_{w \in \mathcal{W}_+} p(w \mid S_i, \text{Probing prompt}),$$

where $p(\cdot|\cdot)$ denotes the continuation probabilities determined by π_θ. We then calculate the average $\hat{c}_i$ after the First Answer Step (FAS) is generated, including the subsequent five reflection steps (Ref1-5). As shown in Figure 4b, the model exhibits relatively low confidence at the FAS, but increases sharply after the first reflection, and continues to rise through subsequent reflections. Notably, when the reasoning chain stops, the average confidence is 0.95, motivating the introduction of a manually controlled lower threshold t_e. We ultimately set $t_e = 0.5$ through experiments.

Additionally, the application of the threshold is carefully controlled to prevent premature termination: once the detected confidence $\hat{c}_i$ exceeds t_e, the model is first prompted to output a final answer (e.g., by appending `Final Answer:`). The reasoning process halts only if this answer is subsequently verified as correct; otherwise, generation continues. More details regarding the selection of t_e and *Early Stopping* are provided in Appendix A.2.

3.4 Building Efficient Reasoning Chains

CONCISE integrates *Confidence Injection* and *Early Stopping* to dynamically control the reasoning generation process. Given an input question, the LRM generates the reasoning chain step-by-step. At each step s_i: if it is identified as a reflection step, *Confidence Injection* is applied by appending a confidence prompt and regenerating s_i with boosted internal confidence. s_i is then added to the current chain S_i. Subsequently, a lightweight confidence detector estimates the detected confidence score $\hat{c}_i$. If $\hat{c}_i > t_e$, the LRM is prompted to provide a final answer a. Crucially, the generation process terminates early at this stage if, and only if, this answer a matches the ground truth gt and $\hat{c}_i > t_e$, otherwise reasoning continues to the next step. After the loop concludes, should a correct final answer a have been verified, a concluding summary (e.g., prompted by `</think>`) is generated and appended to the accumulated steps S_i to form the complete CONCISE reasoning chain S,

---

Algorithm 1: Workflow of CONCISE.

Preparation: LRM generation policy: π_θ, Confidence Phrase Pool: $\mathcal{P}$, Early Stopping Threshold t_e, Prompt Template T
Input: Question q, Ground Truth gt
Initialize: Reasoning chain $S_0 \leftarrow T(q)$
for *each reasoning step* $i = 1, 2, \ldots$ **do**
  $s_i \leftarrow \pi_\theta(S_{i-1})$
  if s_i *is a reflection step* **then**
    Sample a confidence phrase $p_i \in \mathcal{P}$
    $s_i \leftarrow \pi_\theta(S_{i-1}, p_i)$
  $S_i \leftarrow S_{i-1} + s_i$
  Compute detected confidence $\hat{c}_i$
  if $\hat{c}_i > t_e$ **then**
    $a \leftarrow \pi_\theta(S_i, \text{'Final Answer:'})$
    if *isequal*(a, gt) **then**
      break

if *isequal*(a, gt) **then**
  Summary: $S \leftarrow S_i + \pi_\theta(S_i, \text{'</think>'})$
  Output: CONCISE reasoning chain S
else
  S_i and q are **discarded**

---

otherwise, if a wrong answer was finally achieved, the partial chain and the original question are discarded. Algorithm 1 summarizes this coordinated reasoning process in CONCISE.

3.5 Training Objective and Fine-tuning

To align LRM generation with the behavior encouraged by CONCISE, we first construct a fine-tuning dataset by applying the CONCISE pipeline to a set of questions. This yields concise reasoning chains in which redundant reflections are actively suppressed. We then fine-tune the LRM on this dataset using two standard learning paradigms: Supervised Fine-Tuning (SFT) and Simple Preference Optimization (SimPO). Both approaches optimize a shared objective that governs the desired generation behavior:

$$\begin{cases} \pi_\theta(S_i) \rightarrow \pi_\theta(S_i + p_i), & \text{if } c_i < t_i \\ \pi_\theta(S_i) \rightarrow \text{Terminate}, & \text{if } i \geq i' \text{ and } \hat{c}_i > t_e \end{cases}$$

Specifically, when the model's confidence c_i at a given step falls below the threshold t_i, it learns to approximate the distribution conditioned on an injected confidence phrase p_i, which helps improve internal certainty and suppress unnecessary reflections. After the first answer is generated, if the

Model	Math-500			GSM8K			AIME24			GPQA			Average	
	Acc.↑	Tok.↓	CR↓	Acc.↑	Tok.↓	CR↓	Acc.↑	Tok.↓	CR↓	Acc.↑	Tok.↓	CR↓	Acc.↑	CR↓
DeepSeek-7B$_{\text{Origin}}$	90.8	3854	100%	93.1	1442	100%	54.2	13574	100%	51.0	8142	100%	72.3	100%
OverThink$_{\text{SFT}}$	92.2	2538	66%	93.0	1002	70%	52.5	11225	83%	51.1	7639	94%	**72.2**	78%
Spirit$_{\text{SFT}}$	91.0	2935	76%	91.4	1107	77%	51.7	11529	85%	50.3	7084	87%	71.1	81%
ConCISE$_{\text{SFT}}$	92.0	2244	58%	92.9	832	58%	52.1	9751	72%	50.0	5892	72%	71.8	**65%**
OverThink$_{\text{SimPO}}$	91.4	2405	62%	92.9	879	61%	50.0	9603	71%	49.9	6305	77%	**71.0**	68%
Spirit$_{\text{SimPO}}$	87.2	1765	46%	90.8	688	48%	38.3	6926	51%	50.2	5832	72%	66.6	**54%**
ConCISE$_{\text{SimPO}}$	91.0	1946	51%	92.1	715	50%	48.3	7745	57%	48.0	4859	60%	70.0	**54%**
DeepSeek-1.5B$_{\text{Origin}}$	82.2	4784	100%	85.4	2219	100%	29.2	17465	100%	34.5	9492	100%	57.8	100%
OverThink$_{\text{SFT}}$	84.0	3296	69%	85.2	1200	54%	30.0	12893	74%	34.8	8600	91%	**58.5**	72%
Spirit$_{\text{SFT}}$	83.8	3857	81%	84.8	1476	67%	28.8	13685	78%	32.3	8555	90%	57.4	79%
ConCISE$_{\text{SFT}}$	83.6	2701	57%	84.9	923	42%	30.0	11359	65%	35.0	7253	76%	58.4	**60%**
OverThink$_{\text{SimPO}}$	83.6	2738	57%	84.8	1003	45%	31.3	11465	66%	34.7	8250	87%	**58.6**	64%
Spirit$_{\text{SimPO}}$	82.6	2455	51%	82.6	804	36%	26.7	9946	57%	33.1	6910	73%	56.2	**54%**
ConCISE$_{\text{SimPO}}$	83.6	2429	51%	84.3	803	36%	30.4	8810	50%	35.4	7056	74%	58.4	**53%**
Skywork-7B$_{\text{Origin}}$	93.6	4178	100%	93.2	2111	100%	62.9	12464	100%	51.1	8374	100%	75.2	100%
OverThink$_{\text{SFT}}$	92.8	3410	82%	93.1	1561	74%	59.2	11047	89%	50.7	8128	97%	73.9	85%
Spirit$_{\text{SFT}}$	93.4	3279	79%	93.1	1452	69%	58.8	11815	95%	52.0	7565	90%	**74.2**	83%
ConCISE$_{\text{SFT}}$	93.2	2740	66%	92.7	1247	59%	59.2	9871	79%	51.8	6543	78%	**74.2**	**71%**
OverThink$_{\text{SimPO}}$	93.8	2758	66%	93.4	1114	53%	56.3	9422	76%	51.5	6773	81%	73.7	69%
Spirit$_{\text{SimPO}}$	92.0	2329	56%	92.3	801	38%	52.5	8055	65%	51.0	6663	80%	72.0	**59%**
ConCISE$_{\text{SimPO}}$	93.0	2207	53%	93.3	882	42%	55.9	7598	61%	51.7	5668	68%	**74.0**	56%
Qwen3-8B$_{\text{Origin}}$	93.4	5142	100%	95.6	2211	100%	73.3	15094	100%	60.6	7309	100%	80.7	100%
OverThink$_{\text{SFT}}$	93.2	4963	97%	95.5	2133	96%	72.5	14123	94%	60.3	7204	99%	80.4	96%
Spirit$_{\text{SFT}}$	93.2	4184	81%	95.2	1675	76%	73.3	13854	92%	60.0	6784	93%	80.4	**85%**
ConCISE$_{\text{SFT}}$	93.0	4712	92%	95.6	1907	86%	72.1	14168	94%	60.4	6952	95%	**80.6**	92%
OverThink$_{\text{SimPO}}$	93.6	2488	48%	95.2	929	42%	69.2	9468	63%	57.1	3524	48%	78.9	**50%**
Spirit$_{\text{SimPO}}$	93.2	2920	57%	95.3	997	45%	67.9	10441	69%	56.8	3473	48%	78.3	55%
ConCISE$_{\text{SimPO}}$	93.0	2271	44%	95.5	841	38%	71.6	10098	67%	57.6	3751	51%	**79.4**	**50%**

Table 1: Comparison of OverThink, SPIRIT, and ConCISE methods, fine-tuned with SFT or SimPO on four LRMs. Metrics include Accuracy (Acc.), Token Count (Tok.), and Compression Ratio (CR) over four benchmarks. Light gray rows denote original LRM performance. Best and second-best results are in **bold** and underlined respectively.

detected confidence score $\hat{c}_i$ exceeds the early stopping threshold t_e, the model is explicitly trained to terminate reasoning at that point. Through fine-tuning, the model could learn to regulate its generation trajectory based on confidence, strengthening certainty during reasoning and halting once sufficient confidence is reached.

4 Experiments

4.1 Settings

Models and Datasets. We evaluate ConCISE on four LRMs: DeepSeek-R1-Distill-Qwen-7B & 1.5B(Guo et al., 2025), Skywork-OR1-7B-Preview(He et al., 2025), and Qwen3-8B(Team, 2025). As for the training dataset, we select 2,000 questions from the MATH training set(Hendrycks et al., 2021), ensuring that each question yields a correct answer under greedy decoding and maintaining diversity in difficulty and response lengths.

Evaluation. We evaluate model performance across four benchmarks: GSM8K(Cobbe et al., 2021), Math-500(Hendrycks et al., 2021), AIME24, and GPQA_diamond(Rein et al., 2024). All eval-

uations use the same decoding configuration with temperature = 0.6 and top_p = 0.95(Guo et al., 2025). The maximum length is set to 16k for GSM8k, Math-500, GPQA_diamond, and 32k for AIME24. For AIME24 and GPQA_diamond, due to their higher difficulty and smaller sizes, we sample 8 times and report the mean values. As for metrics, we adopt three primary metrics to assess both reasoning accuracy and compression effectiveness comprehensively. Acc. denotes the accuracy of the final answer. Tok. refers to the average response length, measured in tokens. CR (Compression Rate) is defined as the ratio of the average response length to that of the original model, with lower values indicating better compression.

Baselines. We compare our method against existing approaches that aim to remove redundant reasoning and construct efficient reasoning datasets, with further comparisons after fine-tuning under SFT and SimPO scenarios. **OverThink** (Chen et al., 2024) samples eight responses from the LRM with a relatively high temperature and selects the shortest correct one. Only the first reflection step

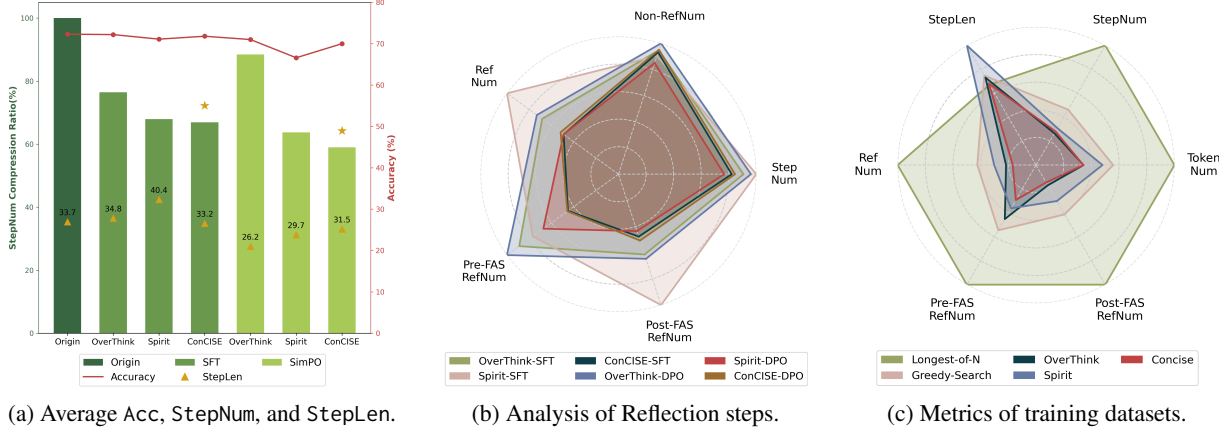| (a) Average `Acc`, `StepNum`, and `StepLen`. | (b) Analysis of Reflection steps. | (c) Metrics of training datasets. |

Figure 5: Further analysis of reasoning chains and training datasets on DeepSeek-R1-Distill-Qwen-7B.

after the answer is retained. **Spirit** (Cui et al., 2025) iteratively removes steps with minimal impact on perplexity (PPL), deleting 30% of the least critical steps. Adjacent contexts are merged to maintain coherence. Appendix B.2 shows more implementation and training details.

4.2 Reults and Analysis

This section evaluates different compression methods across various reasoning benchmarks. Our results show that CONCISE strikes a superior balance between compression and task performance compared to baseline methods, enabling models to effectively eliminate redundant reasoning steps. Additionally, models fine-tuned with CONCISE on mathematical datasets generalize well to GPQA_diamond, showcasing its robustness.

CONCISE achieves superior balance between compression and task performance As shown in Table 1, CONCISE consistently demonstrates strong compression ability across all four LRMs and under both SFT and SimPO settings, while maintaining model performance. Particularly under SimPO settings, CONCISE achieves a compression rate of ~50% with minimal impact on the original model's performance, achieving excellent compression results on the challenging task AIME24 as well as the out-of-domain task GPQA.

In contrast, OverThink achieves competitive task performance but exhibits weaker compression under both SFT and SimPO settings. Regarding Spirit, despite attaining a compression rate comparable to that of CONCISE under SimPO settings, its task performance suffered a notable degradation, especially on challenging tasks like AIME24. Overall, compared to baseline methods, CONCISE offers a better balance between compression and perfor-

mance, demonstrating excellent in-domain and out-of-domain generalization and robustness.

How does CONCISE achieve a better compression & performance balance? To better understand how CONCISE achieves an improved balance between compression and performance, we particularly analyze the structure of reasoning chains generated on various benchmarks of DeepSeek-R1-Distill-Qwen-7B. As shown in Figure 5a, we evaluate the average number of steps (`StepNum`) and the average token count per step (`StepLen`). CONCISE generates the fewest reasoning steps, while its `StepLen` remains comparable to that of the original model.

More detailed analysis focuses on reflection steps within reasoning chains on MATH-500, revealing distinct compression strategies among the methods. As shown in Figure 5b, we find that **pruning reflection steps is key to compression, and preserving non-reflection steps is paramount for model performance.** CONCISE uniquely achieves this critical balance. Specifically, Spirit-SFT retains the most reflection steps (`RefNum`), followed by OverThink, while CONCISE retains the fewest. Regarding non-reflection steps (`Non-RefNum`), OverThink and CONCISE exhibit similar counts. Spirit shows slightly fewer non-reflection steps under SimPO, albeit at the cost of a performance drop. This suggests that current compression techniques primarily achieve compression by pruning reflection steps, as none of the evaluated methods can effectively reduce non-reflection steps without impairing model performance.

Further dissecting the reflection steps, Over-Think and Spirit have the most reflection steps before FAS(`Pre-FAS RefNum`) due to ineffective early-stage reasoning control. In contrast, Spirit-

SFT has most reflection steps after FAS(`Post-FAS RefNum`), indicating its inability to prune redundant post-answer reflections. These distinct strategies in reflection management contribute to performance disparities: OverThink's limited pre-answer pruning curtails its compression potential, while Spirit's difficulty in accurately identifying redundant steps often degrades performance by removing essential content. CONCISE, however, achieves a better compression-performance balance by precisely eliminating redundant reflection steps throughout the entire reasoning chain without compromising the essential reasoning process. We present more details about the above analysis in Appendix B

How does the training dataset affect compression? Since all methods operate under identical training configurations, variations in final compression performance are fundamentally rooted in the **characteristics of their respective training datasets**. To this end, we analyze training datasets constructed by CONCISE, OverThink, Spirit, Greedy-Search (derived from Spirit's rejected samples), and Longest-of-N (derived from rejected samples of CONCISE and OverThink).

As shown in Figure 5c, surface-level metrics such as response tokens (`Tok.`), step count (`StepNum`), and average step length (`StepLen`) show no strong direct correlation with the final compression rate. For instance, while the training datasets of CONCISE and OverThink yield nearly identical values for these metrics, CONCISE demonstrates significantly better compression after fine-tuning under both SFT and SimPO. Conversely, Metrics related to reflection offer more insightful distinctions among the methods. For instance, OverThink has the highest values for `Pre-FAS RefNum`, while Spirit shows the highest `Post-FAS RefNum`. CONCISE, in contrast, consistently maintains lower figures for both these metrics, underscoring its more effective control over the model's reflective tendencies. Echoing the analysis from the previous subsection, these observed characteristics within the training datasets directly mirror the fine-tuned model's capacity for reflection management. This provides strong evidence that the model effectively learns features about reflection from the training data during the fine-tuning process. Thus, it becomes clear how CONCISE uses its unique training data design to achieve significant compression by identifying and cutting down on unnecessary reflection steps.

Benchmark	Method	Acc.	Tok.	StepNum	StepLen	CR
Math-500	Origin	90.8	3854	113.7	33.9	100%
	ConCISE	<u>92.0</u>	**2244.3**	63.0	35.6	**58%**
	Conf. Inj.	91.8	<u>2620</u>	79.5	33.0	<u>68%</u>
	Early Stop	**92.4**	2654	68.4	38.8	69%
GSM8K	Origin	93.1	1442	41.4	34.8	100%
	ConCISE	**92.9**	**832**	22.5	37.1	**58%**
	Conf. Inj.	92.6	<u>934</u>	23.2	40.2	<u>65%</u>
	Early Stop	<u>92.7</u>	1003	22.3	45.0	70%
AIME24	Origin	54.2	13574	463.0	29.3	100%
	ConCISE	<u>52.1</u>	**9751**	333.0	29.3	**72%**
	Conf. Inj.	51.3	<u>10166</u>	372.2	27.3	<u>75%</u>
	Early Stop	**54.2**	12205	389.7	31.3	90%

Table 2: Ablation study of Confidence Injection and Early Stopping on DeepSeek-R1-Distill-Qwen-7B.

4.3 Ablation Study

We conduct an ablation study to evaluate the two core components of CONCISE: *Confidence Injection* and *Early Stopping*. For each, we construct a training dataset and fine-tune DeepSeek-7B using the same configurations as in the main experiments. All experiments are performed under SFT settings and evaluated across three mathematical reasoning tasks: Math-500, GSM8K, and AIME24.

The results, summarized in Table 2, show that both mechanisms achieve similar reasoning accuracy to full CONCISE but exhibit weaker compression performance. Specifically, *Confidence Injection* suppresses unnecessary reflections by boosting the model's confidence, but has a limited impact on terminating reasoning after the final answer. *Early Stopping*, on the other hand, halts excessive post-answer reasoning but does not address earlier stages. Only by combining both can we achieve comprehensive redundancy elimination without compromising reasoning quality.

5 Conclusion

In this work, we first modeled the generation of reflection steps in LRMs from a confidence-guided perspective, helping to understand two key patterns responsible for redundant reflection within the reasoning process of LRMs: *Confidence Deficit* and *Termination Delay*. Based on this, we propose CONCISE, which employs *Confidence Injection* and *Early Stopping* mechanisms to respectively address the above two patterns to suppress redundant reflection steps and produce efficient, concise reasoning chains. Extensive experiments demonstrate that LRMs fine-tuned on CONCISE-generated data significantly compress their responses while maintaining strong task performance.

8

Limitations

While CONCISE demonstrates strong performance in compressing reasoning chains, it is subject to certain limitations that merit further exploration:

Room for Further Compression. While CONCISE effectively addresses redundant reflection steps, analysis indicates that significant potential remains for further compression of both non-reflection steps and the average length of each step. Future work could explore integrating techniques with CONCISE to achieve a higher degree of compression while maintaining model performance.

Confidence Estimation During Reasoning. Although CONCISE introduces a lightweight confidence detector for the post-answer phase, it still lacks a direct mechanism to model confidence before the first answer. Instead, it relies on the occurrence of reflection steps as an indirect proxy. Future work could explore training a lightweight model to detect the model's internal confidence, thereby enabling more fine-grained control.

Inference-Time Compression. CONCISE can also serve as a training-free, decoding-based method that generates concise reasoning chains by intervening in the decoding process. However, because *Confidence Injection* and *Early Stopping* mechanisms add inference overhead, CONCISE lacks a speed advantage over original long reasoning chains, still requiring further optimization.

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*.

Daman Arora and Andrea Zanette. 2025. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*.

Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2024. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

Yu-Neng Chuang, Leisheng Yu, Guanchu Wang, Lizhe Zhang, Zirui Liu, Xuanting Cai, Yang Sui, Vladimir Braverman, and Xia Hu. 2025. Confident or seek stronger: Exploring uncertainty-based on-device llm routing from benchmarking to generalization. *arXiv preprint arXiv:2502.04428*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, and 1 others. 2025. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. *arXiv preprint arXiv:2502.13260*.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2024. Token-budget-aware llm reasoning. *arXiv preprint arXiv:2412.18547*.

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

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. 2025. Skywork open reasoner series. https://capricious-hydrogen-41c.notion.site/Skywork-Open-Reaonser-Series-1d0bc9ae823a80459b46c149e4f51680. Notion Blog.

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. *NeurIPS*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Ayeong Lee, Ethan Che, and Tianyi Peng. 2025. How well do llms compress their own chain-of-thought? a token complexity approach. *arXiv preprint arXiv:2503.01141*.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *arXiv preprint arXiv:2501.12570*.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. 2025. Cot-valve: Length-compressible chain-of-thought tuning. *arXiv preprint arXiv:2502.09601*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*.

Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. 2024. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*.

Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, and 1 others. 2025. A survey of efficient reasoning for large reasoning models: Language, multimodality, and beyond. *arXiv preprint arXiv:2503.21614*.

Qwen Team. 2024. QwQ: Reflect Deeply on the Boundaries of the Unknown. Accessed: 2025-04-05.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Matthew Renze and Erhan Guven. 2024. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pages 476–483. IEEE.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, and Shiguo Lian. 2025a. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *arXiv preprint arXiv:2503.04472*.

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

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Hanjie Chen, Xia Hu, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

Hanshi Sun, Momin Haider, Ruiqi Zhang, Huitao Yang, Jiahao Qiu, Ming Yin, Mengdi Wang, Peter Bartlett, and Andrea Zanette. 2024. Fast best-of-n decoding via speculative rejection. *arXiv preprint arXiv:2410.20290*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Qwen Team. 2025. Qwen3.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, and 1 others. 2025. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266*.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping Wang. 2025. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. Lightthinker: Thinking step-by-step compression. *arXiv preprint arXiv:2502.15589*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

A Details about ConCISE

A.1 Confidence Injection

The selection of appropriate confidence phrases is critical for the Confidence Injection component of ConCISE. We curated a pool of 20 distinct phrases, presented in Table 3, by observing the model's native reasoning expressions, employing manual design, and performing experimental refinement (partly illustrated in Figure 4a). The purpose of these phrases is to inject confidence into the model, thereby preventing redundant reflection steps during reasoning. To mitigate potential overfitting to any specific phrase, one phrase is randomly selected from the pool for injection at runtime.

Additionally, another crucial mechanism within the Confidence Injection process is identifying whether the current step constitutes a reflection step. The accurate and swift identification of such steps is paramount for the success of Confidence Injection. Consequently, informed by our observations of reflection steps during LRM reasoning processes, we employed a rule-based detection method. This method, which is detailed in Appendix A.3.1, achieves reasonably accurate detection while having a negligible impact on the model's inference speed.

A.2 Early Stopping

Probing prompt and confidence-indicated tokens. The effectiveness of the Early Stopping mechanism relies heavily on the accuracy of its confidence detector. To identify an appropriate probing prompt, we first analyzed the reasoning chains generated by four Large Reasoning Models (LRMs) using Greedy decoding on the training question set. This analysis revealed that the model frequently expresses confidence during the later part of the reasoning chain, and as shown in Table 4, self-reported confidence expressions starting with "I'm" (such as "I'm confident") are very common.

Based on this observation, particularly considering the prevalence of confidence statements led by "I'm", we considered "So, I'm" as a potential probing prompt. Concurrently, we also experimented with alternative probing prompts, such as "So, I feel" and "I can be". However, we observed that these alternative prompts often resulted in high confidence scores before the model had reached the final answer (i.e., prior to Final Answer Serialization, FAS), leading to unnecessary computational

Table 3: The Pool of 20 Confidence Phrases used for Confidence Injection.

Index	Confidence Phrase
1	Therefore
2	The reasoning holds
3	Previous steps are correct
4	All steps are valid
5	With this established
6	That sounds reasonable
7	Let's go ahead
8	Alright, let's carry on
9	Let's proceed
10	Let's progress
11	So, putting it all together
12	The logic stands firm
13	The reasoning process is valid
14	Good, let's keep going
15	Everything seems reasonable so far
16	This part checks out
17	I think that's solid. So
18	The reasoning holds, let's keep going
19	Everything checks out, let's move on
20	All steps are solid, let's move forward

overhead.

Therefore, considering these factors—namely, the prevalence of "I'm"-led confidence statements in the model's natural expressions and the shortcomings of the alternative prompts—we ultimately selected "So, I'm" as the fixed probing prompt. Subsequently, we identified the primary confidence-indicative tokens that follow the "So, I'm" prompt, which are: "confident", "pretty confident", "sure", and "pretty sure".

The confidence score, $\hat{c}_i$, for a given state S_i is then calculated by summing the probabilities of generating these tokens or sequences immediately following the probing prompt "So, I'm":

$$
\begin{aligned}
\hat{c}_i =& P(\text{"confident"} \mid S_i, \text{"So, I'm"}) \\
&+ P(\text{"sure"} \mid S_i, \text{"So, I'm"}) \\
&+ P(\text{"pretty"} \mid S_i, \text{"So, I'm"}) \times \\
&\quad \big(P(\text{"confident"} \mid S_i, \text{"So, I'm pretty"}) \\
&\quad\quad + P(\text{"sure"} \mid S_i, \text{"So, I'm pretty"}) \big)
\end{aligned}
$$

Here, notation like $P(\text{token} \mid S_i, \text{sequence})$ represents the conditional probability of generating the next token given the preceding state S_i and the specific text sequence provided as context. As demonstrated in Figure 4b and supported by our exper-

Table 4: Frequency of Common Confidence Expressions Observed in 2000 training Reasoning Chains Across Different Large Reasoning Model Series(DeepSeek, Qwen and Skywork)

Expression	DeepSeek-7B	DeepSeek-1.5B	Qwen3-8B	Skywork-OR17B
I'm confident	423	335	127	560
I feel confident	170	172	58	144
I can be confident that	157	156	475	130
I can confidently say	70	176	91	98
This gives me (more)confidence that	68	81	23	86
I'm pretty confident	43	69	2	60
I'm pretty sure	37	28	2	17
Therefore, confident that	1	0	26	5
Total (sum of listed expressions)	**969**	**1017**	**804**	**1100**

imental results, the confidence detector designed using this probing prompt and set of confidence-indicative tokens proves effective for the early stopping mechanism in ConCISE.

Early stopping threshold. The selection of the early stopping threshold, t, was guided by the principle of tailoring the reflection process to the apparent quality of the reasoning chain. Our objective was to configure the threshold such that:

- Simple problems with rigorous reasoning steps require no post-answer reflection.

- Problems of moderate difficulty benefit from exactly one round of reflection.

- Difficult problems or those with less rigorous reasoning undergo two or more rounds of reflection.

To find a threshold value that aligns with this principle, we empirically evaluated $t_e \in \{0.4, 0.5, 0.6, 0.7\}$ on the DeepSeek-R1-Distill-Qwen-7B model. For each value, we measured the probability distribution of the exceeding points(when $\hat{c}_i > t_e$) across different stages: exceeding after the initial answer generation (0 reflections), after the first reflection, after the second reflection, etc. The results of this analysis are illustrated in Figure 6.

Observing the distributions in Figure 6, we found that these t_e settings generally cause the model to achieve a confidence score $\hat{c}_i > t_e$ at the position of the first reflection, which aligns with our three guiding principles. Furthermore, the confidence monitored by our designed lightweight detector rarely indicates high confidence before the FAS
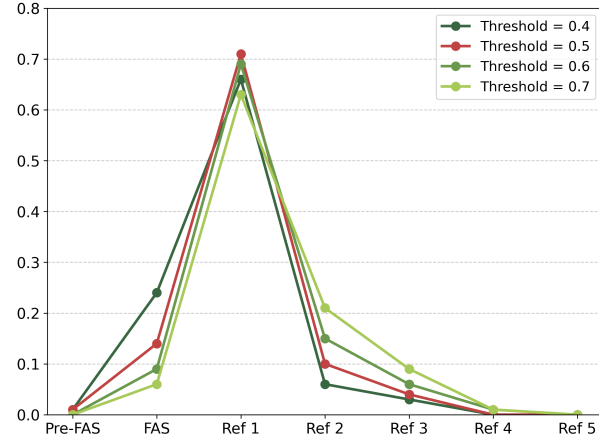

Figure 6: Distribution of exceeding points(when $\hat{c}_i > t_e$) probability of Pre-FAS, FAS, Ref1, 2, ..., 5 for different threshold values (t_e).

Benchmark	T_e	Acc.	Tok.	StepNum	StepLen	CR
Math-500	0.4	91.4	2390.6	68.4	34.9	62.0%
	0.5	**92.0**	**2244.3**	**63.0**	**35.6**	**58.2%**
	0.6	91.8	2324.3	66.1	35.2	60.3%
	0.7	92.0	2345.1	67.5	34.8	60.8%
GSM8K	0.4	92.4	828.9	19.9	41.6	57.5%
	0.5	**92.9**	**831.9**	**22.5**	**37.1**	**57.7%**
	0.6	93.0	831.7	20.4	40.7	57.7%
	0.7	92.7	849.3	21.7	39.2	58.9%
AIME24	0.4	45.8	11548.7	402.1	28.7	85.1%
	0.5	**52.1**	**9750.8**	**333.0**	**29.3**	**71.8%**
	0.6	52.1	9802.5	340.5	28.8	72.2%
	0.7	51.7	9719.9	339.8	28.6	71.6%

Table 5: Effect of different early stopping thresholds (0.4, 0.5, 0.6, 0.7) on the SFT performance of DeepSeek-R1-Distill-Qwen-7B across mathematical benchmarks.

13

(Final Answer Serialization) step. This is advantageous as it effectively reduces the computational overhead associated with prematurely judging the correctness of an answer, making it highly compatible with the ConCISE methodology.

Therefore, we further conducted experiments for $t_e \in \{0.4, 0.5, 0.6, 0.7\}$ under the SFT setting of DeepSeek-R1-Distill-Qwen-7B, and Table 5 presents these experimental results. We found that under the settings of $t_e = 0.5, 0.6$, and 0.7, the model ultimately exhibited similar performance in both compression rate (CR) and accuracy (Acc), demonstrating robustness to the choice of t_e. Consequently, we selected $t_e = 0.5$ for subsequent extensive experiments. However, when $t_e = 0.4$, there was a significant decline in performance. This is because when t_e is set to 0.4, more data exists at the FAS step, preventing the model from adequately checking its reasoning process; this excessive confidence leads to a performance drop.

A.3 Reflection step detection

The detection of reflection steps, as mentioned throughout this paper, plays a crucial role in both the algorithmic flow and experimental evaluation of ConCISE. Specifically, we employ two distinct methods for identifying these steps, selecting the most suitable one depending on the specific application context. These methods and their corresponding usage scenarios are detailed below.

A.3.1 Rule-based detection.

Reflection steps generated by the model often exhibit distinct linguistic characteristics, frequently containing specific keywords (e.g., ″wait″, ″alternatively″) that signal potential reasoning shifts or reconsiderations. Consequently, a straightforward and effective strategy for identifying the start of such steps is rule-based detection. This method checks for the presence of predefined "reflection keywords" within the generated text of a given step, S_i.

Formally, let K be the predefined set of reflection keywords. A step s_i is identified as the start of a reflection, denoted as IsReflectionStart(s_i), if its text contains any keyword k from the set K:

$$\text{IsReflectionStart}(s_i) \equiv \bigvee_{k \in K} (k \subseteq s_i)$$

where $k \subseteq s_i$ indicates that the keyword k appears as a substring of step s_i. The set K of reflection keywords used for rule-based detection consists of the following:

- ″wait″, ″alternatively″, ″check″, ″reconsider″, ″reflect″, ″rethink″, ″reconsidering″, ″reviewing″, ″reassess″, ″pause″, ″second thought″, ″reevaluate″, ″verify″, ″think again″.

While effective for identifying the onset of a reflection sequence, this method typically cannot recognize subsequent steps within the same reflection process, as these later reflection steps often lack the initial triggering keywords. Despite this limitation, we utilize this rule-based approach within the **Confidence Injection**. Since Confidence Injection operates during the model's generation process, identifying and potentially preventing only the first redundant reflection step is often sufficient to avoid generating the entire reflection. This method is particularly advantageous here due to its efficiency (introducing no computational overhead) and ease of implementation.

Similarly, this rule-based detection is used for the Figure 4a analysis to determine if an intervention (like inserting a confidence phrase) triggers the start of a new reflection, as only detecting the initial step with this lightweight method is necessary.

A.3.2 LLM-as-a-Judge

When the task requires identifying all reflection steps within an entire reflection behavior—a capability beyond the rule-based method—a more comprehensive approach is needed. For example, implementing the OverThink baseline necessitates isolating and retaining the complete first reflection sequence that occurs after the FAS(First Answer Step). To address this, we employ an LLM-as-a-Judge methodology.

Specifically, we utilize the Qwen-Max model as the judge. We provide it with both the original problem/question and the model's complete reasoning chain as input. The prompt supplied to Qwen-Max includes precise definitions of the FAS and the characteristics defining a 'reflection behavior'. The model is instructed to return a structured output that annotates the input chain, identifying the index of the FAS and providing the indices for all detected reflection steps, grouped according to the reflection behavior they belong to. We then parse this structured output to finalize the detection and grouping of reflection steps. Illustrative examples of the prompt structure, input format, and expected

14
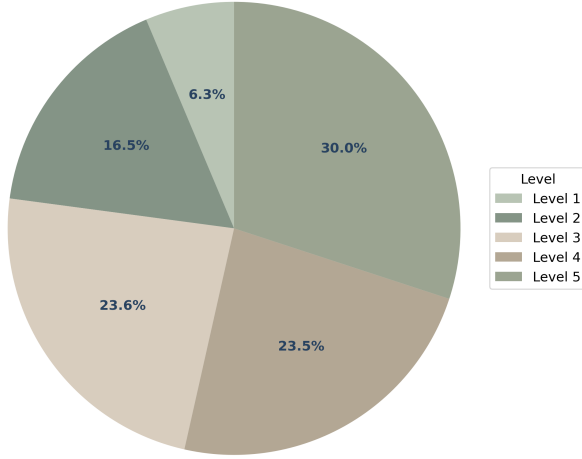
(a) Difficulty diversity of question set.
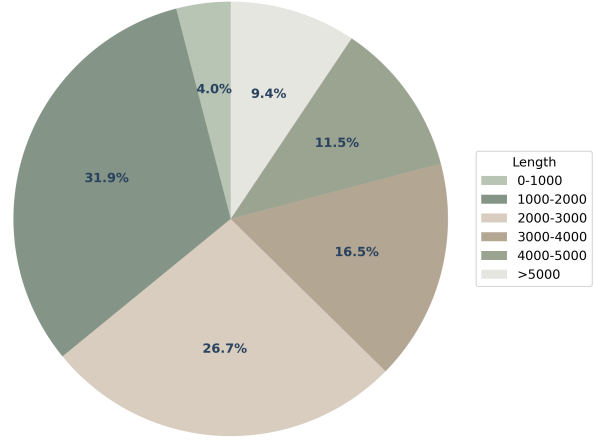
(b) Length diversity of question set.

Figure 7: Diversity of the question set, showing distributions for difficulty and length.

response can be found in Figure 9, Figure 10, and Figure 11

Our observations indicate that Qwen-Max performs reliably in identifying FAS and grouping reflection steps when the reasoning chains are of moderate length. However, we noted a decline in performance for chains exceeding 5000 tokens. This degradation is likely attributable to the known limitations of current LLMs in processing very long contexts effectively. Consequently, for the implementation of the OverThink baseline comparison, reasoning chains longer than 3000 tokens were manually annotated to ensure accuracy.

Furthermore, the LLM-as-a-Judge approach is the basis for calculating several reflection-based metrics presented in our results, such as RefNum, Non-RefNum, pre-FAS RefNum, and post-FAS RefNum (shown in Figure 5b and Figure 5c).

B Details about Experiments

B.1 Datasets

Question Set While the construction methodology for the question set is detailed in Section 4, here we focus on illustrating its diversity in terms of difficulty and length.Figures 7a and 7b demonstrate the diversity of the Question set. Difficulty distribution is derived from metric Level in the MATH dataset, while length distribution reflects reasoning chain lengths from DeepSeek-R1-Distill-Qwen-7B under greedy decoding. Furthermore, it is important to note a subsequent refinement process for these selected questions when constructing the final training set. Although the initial 2000 data entries were chosen based on the premise that the model

could correctly answer them using greedy search, the generation processes for our CONCISE and OverThink methods do not strictly adhere to greedy decoding, and CONCISE will discard wrong reasoning chains. Therefore, for training purposes, we ultimately utilize the common subset of correctly reasoned chains derived from the outputs of three methods: CONCISE, OverThink, and Spirit. This intersection results in a refined training dataset of approximately 1900 samples.

Validation Dataset For the experiments presented in Figure 4, we curated the Verification Dataset. This was necessary because the required analyses (including First Answer Step (FAS) and reflection detection, as shown in Figure 4b) rely on annotations from the LLM-as-a-Judge method (§A.3.2), whose reliability decreases on long reasoning chains. To ensure dependable results, the Verification Dataset comprises another 1000 reasoning chains selected from the MATH training set, filtered based on two criteria: yielding a correct answer under greedy decoding and having a reasoning chain length under 5000. This curated set facilitates reliable LLM-based annotation for the validation experiments.

Evaluation Dataset The analysis presented in Figure 5b and Figure 8b required LLM-as-a-Judge annotations. To ensure reliable results despite this method's limitations with long contexts, we filtered the MATH500 dataset. We selected only those problems where reasoning chains generated by all twelve specified finetuning configurations (combinations of ConCISE/OverThink/Spirit methods, 1.5B/7B sizes, and SFT/SimPO techniques) were

15

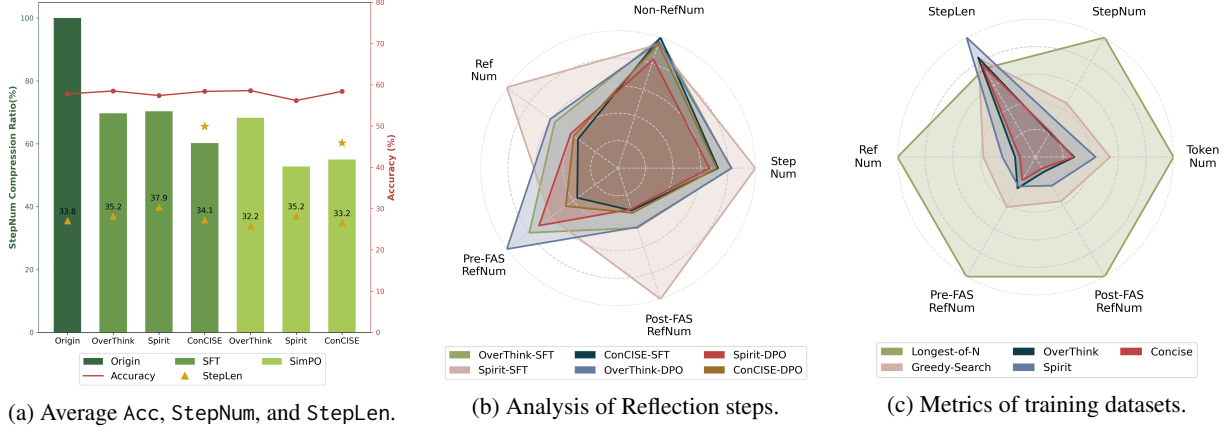| (a) Average Acc, StepNum, and StepLen. | (b) Analysis of Reflection steps. | (c) Metrics of training datasets. |

Figure 8: Further analysis of reasoning chain and training dataset characteristics on DeepSeek-R1-Distill-Qwen-1.5B.

shorter than 5000 tokens. This yielded a common subset of 340 problems, on which our subsequent comparative analysis of the different methods was exclusively performed.

B.2 Training Details

Models. DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Qwen-1.5B are reasoning models derived by DeepSeek from the Qwen2.5 series models (Yang et al., 2024)through distillation training, using training and generation data from the DeepSeek-R1 model. Skywork-OR1-7B-Preview was further developed via reinforcement learning, exhibiting strong capabilities on test sets such as mathematics. Meanwhile, Qwen3-8B is a recently released hybrid reasoning model noted for its powerful reasoning abilities. All four of these are open-source models. While they possess strong Chain-of-Thought (CoT) reasoning capabilities, they also present clear issues with reasoning redundancy. Consequently, we employ these four models for subsequent training and evaluation. Furthermore, we ensure that these models, along with all datasets involved in the training and evaluation processes, are utilized following their original licenses and intended purposes.

Training. We construct training datasets based on the question set using CONCISE, OverThink, and Spirit. For the generation process of Con-CISE, we set Temperature=0.6, Top_p=0.95. For OverThink, we sample 8 reasoning chains under Tempature=1.0(which is relatively high due to the recommended temperature for solving math problems of the LRMs is usually 0.6-0.8), Top_p=0.95. For Spirit, compression is applied to reasoning chains generated via greedy search. During training, we adopt two strategies: SFT and SimPO. SFT directly uses the question and its corresponding compressed reasoning trace for supervised fine-tuning. For SimPO, reject samples are also required (Rafailov et al., 2023). Following the original OverThink setup, we use the longest correct reasoning chain among the eight samples as the reject samples. To ensure a fair comparison, CON-CISE adopts the same strategy. For Spirit, the full reasoning chain before compression serves as reject samples, because this is precisely the comparative effect Spirit aims to demonstrate: important steps are retained while preference relationships are learned only for non-important steps. We use LlamaFactory for both SFT and SimPO fine-tuning (Zheng et al., 2024). All models are trained with $lr = 1e - 6$; SFT runs for 2 epochs and SimPO for 1 epoch. The effective batch size for all training is 32 (using per_device_train_batch_size=1 and gradient_accumulation_steps=8 on 4 A800 GPUs), and max_length is set to 10240.

B.3 Analysis on DeepSeek-1.5B

Regarding the analysis conducted on DeepSeek-R1-Distill-Qwen-7B in the Experimental Section 4.1, we also performed a similar analysis for DeepSeek-R1-Distill-Qwen-1.5B. Figure 8 shows the specific results. The trends exhibited by the data, as well as the relationships among the three methods, are largely consistent with those presented for DeepSeek-R1-Distill-Qwen-7B. Consequently, the final conclusions are also consistent.

You are an AI assistant trained to analyze reasoning steps in a response. Your task has two parts:
1. Examine each reasoning step to determine if it's part of a reflection process.
2. Identify the earliest step where the final answer (as later shown in boxed{}) is first derived, regardless of whether it is formally written or boxed at that moment.

[Definition of Reflection]:
1. A reflection process is a sequence of one or more reasoning steps that recheck or doubt a previously made conclusion, such as double-checking calculations, using alternative methods.
2. Typical signals include (but are not limited to): 'Wait', 'Alternatively', 'Just to double check', 'But hold on', etc. These signals usually mean the start of a new reflection process.
3. However, even without such phrases, if the content of a step reflects a verification or reevaluation, it should be marked as a part of a reflection process.

[Output Format]:
1. **Reflection Step**: List all reflection processes as groups of steps.
    - If Step3 and Step4 form a reflection, write as (Step3, Step4)
    - If Step5, Step6, Step7 form a new reflection process together, list as a separate group: [(Step3, Step4), (Step5, Step6, Step7)]
    - Avoid putting a lot of steps into one single reflection process.
2. **First Answer Step**: Write the earliest step where the final answer is first derived(e.g., Step2).

[Example]:
Question: 2 + 3 = ?
Response: *Step1*: I start with 2 + 3. *Step2*: That gives me 5. *Step3*: Wait, is that right? Let me make sure... *Step4*: But hold on, let me double-check. Maybe I should... *Step5*: Wait, no, I think... *Step6*: Therefore, the result is 5 *Step7*: Alternatively, if I use... *Step8*: So, the answer is 5 *Step9*: **Final Answer**: the answer is boxed5.
Output:
Reflection Step: [(Step3), (Step4), (Step5, Step6), (Step7, Step8)]
First Correct Step: Step2
Explanations for Reflection Step:
    - There is 'wait', a reflection keyword, in Step3, so Step3 is the start of a reflection process.
    - There is 'But hold on' in step4, so step4 means the start of a new reflection process.
    - Then, there is 'wait' in step5, so step5 means the start of a new reflection process.
    - There is 'therefore' in step6, so (step5,step6) is in the same refleciton process.
    - There is 'alternatively' in step7, so step7 is the start of a new reflection process.
    - There is 'so' in step8, so (step7,step8) is in the same reflection process.

[Task]:
Now analyze the following question and response:
f"Question: {question}"
f"Response: {response}"
Please output the reflection steps and the first correct step in the format without extra explanation.

Figure 9: Prompt of using the LLM-as-a-Judge method for reflection step detection.

Figure 10: An example input of using the LLM-as-a-Judge method for reflection step detection.

Figure 11: An example output of using the LLM-as-a-Judge method for reflection step detection.