Slim-SC: Thought Pruning for Efficient Scaling with Self-Consistency

Colin Hong[1,*], **Xu Guo**[2,*✉], **Anand Chaanan Singh**[1],
Esha Choukse[3], **Dmitrii Ustiugov**[1]
[1]NTU Singapore [2]KTH Royal Institute of Technology [3]Microsoft
{hong0259,xu008}@e.ntu.edu.sg, dmitrii.ustiugov@ntu.edu.sg
[*] Equal contribution. [✉] Corresponding author.

Abstract

Recently, Test-Time Scaling (TTS) has gained increasing attention for improving LLM reasoning performance at test time without retraining the model. A notable TTS technique is Self-Consistency (SC), which generates multiple reasoning chains in parallel and selects the final answer via majority voting. While effective, the order-of-magnitude computational overhead limits its broad deployment. Prior attempts to accelerate SC mainly rely on model-based confidence scores or heuristics with limited empirical support. For the first time, we theoretically and empirically analyze the inefficiencies of SC and reveal actionable opportunities for improvement. Building on these insights, we propose Slim-SC, a step-wise pruning strategy that identifies and removes redundant chains using inter-chain similarity at the thought level. Experiments on three STEM reasoning datasets and two recent LLM architectures show that Slim-SC reduces inference latency and KVC usage by up to 45% and 26%, respectively, with R1-Distill, while maintaining or improving accuracy, thus offering a simple yet efficient TTS alternative for SC.

1 Introduction

In recent years, advances in Large Language Models (LLMs) have mainly come from training-time scaling, through more parameters, bigger datasets, and more compute (Brown et al., 2020; Chowdhery et al., 2023; OpenAI, 2023; Dubey et al., 2024; DeepSeek-AI, 2025a). Such scale has unlocked emergent reasoning abilities, e.g., through Chain-of-Thought (CoT) prompting (Wei et al., 2023), which prompts the LLM to generate intermediate thinking steps to enhance answer accuracy. Building on this foundation, a new scaling paradigm known as Test-Time Scaling (TTS) (Wang et al., 2023; Snell et al., 2025; Brown et al., 2024; Muennighoff et al., 2025) has gained traction, which aims to further enhance reasoning performance by allocating additional computation at inference time, without modifying the model's parameters.

Current TTS methods can generally be divided into two categories. Sequential scaling, such as Least-to-Most prompting (Zhou et al., 2023), directs later thinking steps based on earlier reasoning results (Madaan et al., 2023; Khot et al., 2023). While this can improve local coherence and reasoning depth, it is also vulnerable to error propagation. Parallel scaling, such as Best-of-N (BoN) (Lightman et al., 2024) and Self-Consistency (SC) (Wang et al., 2023), takes a different route. They sample many reasoning chains independently and aggregate the final answer through a fusion mechanism. These methods aim to increase diversity among reasoning paths to improve the chance of reaching a correct answer. However, the extra samples raise non-trivial latency and cost (Fig. 1a).

Recent attempts to speed up parallel scaling mainly focus on sampling fewer chains (Aggarwal et al., 2023; Li et al., 2024; Wang et al., 2025a; Wan et al., 2025): they generate chains in batches, perform an early vote, and stop if the batch already agrees. This strategy shifts part of the computation into a sequential loop, where each new batch must wait for the previous vote, thereby cutting total compute but often increasing end-to-end latency. Moreover, our study shows that correct and incorrect chains often exhibit different patterns (Fig. 1c, Fig. 2b). Early agreement inside any single batch, therefore, does not guarantee accuracy improvement for SC. Results in Tab. 1 confirm this and are consistent with Li et al. (2024).

Some approaches maintain a fixed number of reasoning chains while selectively pruning each chain as it unfolds. They typically rely on a pretrained reward model to estimate confidence at intermediate steps (Sun et al., 2024; Wang et al., 2025b). Others adopt information-theoretic metrics, including entropy trajectory (Guo, 2025), which monitors reasoning utility, and inter-chain distances (Zhou et al.,

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 34489–34506
November 4-9, 2025

2025) to detect outlier chains. However, reward-based pruning heavily depends on the accuracy of the confidence estimation model, which may not always be available. Meanwhile, pruning outliers based on inter-chain distances can be unreliable, as our study shows that correct and incorrect chains each form distinct clusters (Fig. 2b).

To improve the efficiency of Self-Consistency (SC) without compromising accuracy, we propose **Slim-SC**, a step-wise pruning method that removes reasoning chains based on inter-chain similarity at the thought level. Our motivation study shows that **(1)** scaling SC chains indeed increases the presence of accurate answers in the candidate pool, but they are often overwhelmed by a larger number of incorrect ones, leading to the wrong final answer and, hence, lower accuracy (Fig. 2a); **(2)** chains that lead to incorrect answers tend to be longer (Hassid et al., 2025), as compared to the ones leading to correct answers, hence increasing the latency of SC. Since chains with highly similar intermediate thoughts almost always converge to the same final answer, pruning redundant chains before voting yields a triple benefit: lowering compute cost, latency, and improving accuracy when incorrect chains are abundant. Because correct and incorrect chains are well-separated, our similarity-based pruning primarily eliminates redundancy within each cluster, thereby preserving overall accuracy.

Driven by these insights, Slim-SC employs two mechanisms: first, it introduces a warm-up delay, ensuring meaningful pruning, since reasoning chains typically start by problem restatement; second, it applies a similarity-based pruning rule at each thought to decide which chain to retain when two chains become highly similar. We experimentally evaluate both random selection and diversity-based selection for this pruning rule. Extensive experiments on three STEM reasoning datasets and two recent LLM architectures confirm that Slim-SC maintains or improves the accuracy of SC and outperforms existing pruning methods. More importantly, Slim-SC substantially reduces the latency of SC (up to 45%), as well as the generated token number (up to 34%), and KVC usage (up to 26%). Slim-SC is thus a simple yet effective and broadly compatible approach, suitable for integration into various parallel scaling frameworks using SC or similar methods (Li et al., 2024; Lightman et al., 2024; Sun et al., 2024).

Our code is available at `https://github.com/hyscale-lab/slimsc`.

2 Motivation

2.1 Self-Consistency

Given an input x with groundtruth y, an LLM θ under CoT prompting generates a reasoning chain c for x. SC enables test-time scaling by sampling a set of N independent reasoning paths $\mathcal{R}_N = \{c_i\}_{i=1}^{N} \overset{\text{i.i.d.}}{\sim} P_\theta(c \mid x)$ and maps each reasoning chain to an answer $\hat{y}_i = g(c_i)$. The final prediction $\tilde{y}$ is obtained by *plurality voting* (i.e., selecting the most frequent answer):

$$\tilde{y} \;=\; \arg\max_{y} \sum_{i=1}^{N} \mathbf{1}\big[\hat{y}_i = y\big].$$

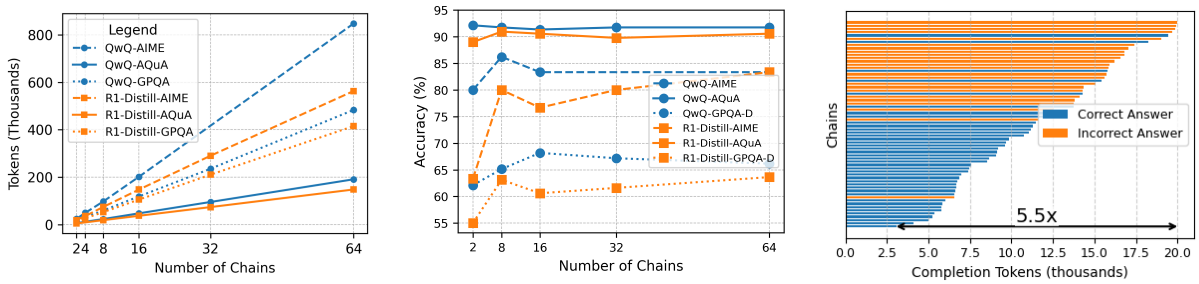
SC improves accuracy only if correct chains occur more often than incorrect chains in the candidate pool. Let p_y be the probability of generating the correct answer y, and $\{p_{y'_j}\}$ be the probabilities for each incorrect answer y'_j. SC is expected to improve accuracy if the correct reasoning path has a probabilistic advantage, i.e., $p_y > \max_j(p_{y'_j})$. Even when $p_y < 0.5$, as long as p_y exceeds every $p_{y'_j}$, increasing N will raise the probability that y wins the plurality vote.

2.2 Motivation Study

To characterize the costs and benefits of standard SC, we investigate its performance across various datasets and models. We aim to quantify how accuracy and computational cost scale with the number of SC chains (N). Furthermore, we analyze the nature of these chains, particularly focusing on their completion lengths and semantic similarity, to identify sources of inefficiency and opportunities for optimization. Specifically, we examine the similarity patterns within and between chains that lead to correct versus incorrect final answers, referred to as *correct* and *incorrect* chains respectively. For this characterization, we primarily use `AIME-2024`, `GPQA Diamond`, and `AQuA-RAT` datasets. The LLMs evaluated are Qwen-QwQ-32B and DeepSeek-R1-Distill-Qwen-14B, which we will shorthand as QwQ and R1-Distill for the rest of the paper.

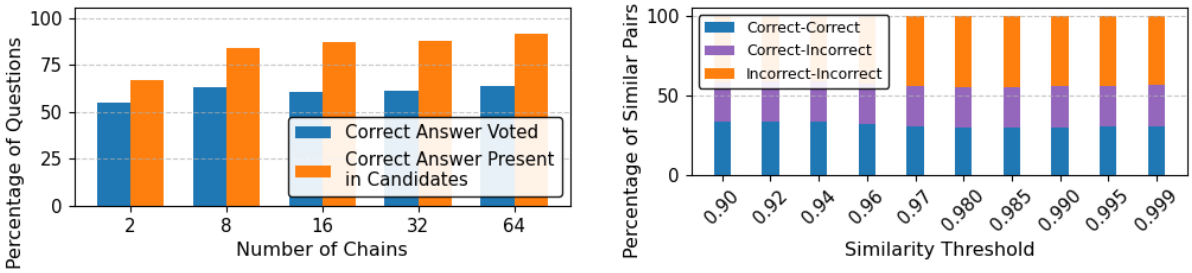
2.3 Problems with Scaling SC Chains

Diminishing returns and massive resource waste in scaling SC chains. SC can substantially improve reasoning accuracy over single-chain CoT. However, this enhanced performance is accompanied by a steep computational cost that accompanies a more gradual accuracy gain as the number



(a) Computational cost scaling. (b) Accuracy across different datasets. (c) SC ($N = 64$) for R1-Distill on AIME.

Figure 1: **Problem with Self-Consistency.** (a) and (b): Diminishing returns from scaling test-time computations. Accuracy generally improves with more chains, but plateaus and diminishes in gains, especially for easier datasets such as AQuA-RAT. (c) SC waits for all chains, including longer incorrect ones. (Appendix A.1 shows the distribution of completion tokens for correct vs. incorrect chains on the AIME dataset. Appendix A.2 gives more examples.)



(a) R1-Distill on GPQA Diamond: % of correct answers appearing in the candidate set vs. being selected as the final answer.

(b) Pairwise similarity of chains using R1-Distill on GPQA Diamond.

Figure 2: **Opportunities.** (a) Many correct chains go unselected, suggesting substantial room for accuracy improvement. (b) Correct and incorrect chains form separate clusters, with high internal similarity within cluster.

of chains increases. As observed in Fig. 1a, the mean total completion tokens generated per question scales linearly with the number of chains (N). For instance, using R1-Distill on the AIME dataset, increasing N from 2 to 64 results in a more than 30-fold increase in token generation. While accuracy generally improves with more chains (Fig. 1b), the gains tend to plateau, especially for easier datasets like AQuA-RAT, making the continued increase in computational cost less justifiable.

Short correct chains have to wait for longer incorrect chains. Standard SC requires generating N complete reasoning paths before majority voting can occur. This *wait-for-all* strategy introduces significant inefficiencies (Hassid et al., 2025). Fig. 1c exemplifies this with R1-Distill on an AIME Q24 problem ($N = 64$): a correct answer might be derived from a short chain, yet the system must wait for all chains, including potentially much longer chains that yield incorrect answers (e.g., a chain can be six times longer than the shortest correct one). Such delays for unhelpful chains contribute to substantial time and computational overhead.

2.4 Opportunities for Improvement

Correct chains can be outvoted by incorrect ones. Even when a correct reasoning path is generated, majority voting can miss it. Fig. 2a (R1-Distill on GPQA Diamond) shows a persistent gap between the *Correct Answer Present in Final Candidates* (Ideal Accuracy) and the *Correct Answer Voted* (Actual Accuracy). For $N = 64$, while the correct answer is present in candidates 91.9% of the time, it is only voted as the final answer 63.6% of the time. This *opportunity gap* suggests that *the presence of noisy or unhelpful chains can mislead the voting process*. Thus, merely increasing the number of chains is insufficient; improving the set of effective candidate chains for voting is crucial.

Correct and incorrect chains are semantically distant. Fig. 2b presents an analysis of the pairwise similarity of chains for R1-Distill on GPQA Diamond with $N = 64$ (computation details in Appendix A.3, and more examples in Fig. 7). Key observations:

- **High Intra-Category Similarity:** Both correct and incorrect chains demonstrate strong internal

similarity. For instance, at the 0.98 threshold, most highly similar pairs are either 'Correct–Correct' (30%) or 'Incorrect–Incorrect' (45%), suggesting that reasoning chains leading to the same outcome tend to follow comparable trajectories.

• **Low Cross-Category Similarity:** Correct and incorrect chains rarely overlap semantically. Only 25% of highly similar pairs are 'Correct–Incorrect', indicating that the two groups form largely distinct clusters in the representation space. As shown in Fig. 7, this proportion is even smaller for both R1-Distill and QwQ models on the AQuA dataset.

• **Increasing Threshold Amplifies the Clustering of Incorrect Chains:** Raising the similarity threshold increases the proportion of similar 'Incorrect-Incorrect' pairs. For instance, at the 0.98 threshold, 45% of highly similar pairs are 'Incorrect-Incorrect', compared to only 30% that are 'Correct-Correct'. As illustrated in Fig. 7, this effect is particularly pronounced for R1-Distill relative to QwQ. These results suggest that flawed reasoning converges more consistently on similar (wrong) logic than valid reasoning does on the same correct logic.

These patterns create an opportunity to prune redundancy without harming diversity. The key insight is that correct and incorrect chains are semantically distant, so a high-similarity pruning strategy will primarily remove chains *within* a category rather than across categories. This means pruning is unlikely to mistakenly remove a correct chain that happens to be similar to an incorrect one. Instead, the primary effect of pruning is removing *redundancy* by eliminating chains that are very similar to others, aiming to preserve a smaller, more diverse set of unique reasoning paths for the final vote.

This opportunity directly targets the multiple inefficiencies of standard SC. The high resource cost and diminishing returns of scaling N (Fig. 1a,b), combined with the severe latency bottleneck of the *wait-for-all* approach (Fig. 1c), underscore the need for a smarter strategy than simply generating more chains. By managing redundancy through semantic pruning, we can create a method that simultaneously reduces computational waste and the *wait-for-all* problem, and potentially improves the quality of the final vote by preserving diversity.

3 The Thought-Pruning Method

Fig. 2b reveals that semantically similar intermediate thought pairs converge on identical outcomes: similar correct chains lead to the same correct answer, while similar incorrect chains lead to incorrect results. This redundancy suggests an opportunity for efficiency: by pruning chains that exhibit highly similar thinking processes to others, we can potentially reduce computational overhead while preserving, or even enhancing, the diversity of unique reasoning paths crucial for accurate decision-making. Our goal is to maintain a compact set of diverse chains, pruning away those that offer little new information, particularly if those similar chains might collectively reinforce an incorrect answer due to a lack of diverse perspectives.

3.1 The Pruning Delay Mechanism

A key consideration in our pruning strategy is the warm-up phase of reasoning. CoT reasoning often begins with common preliminary thinking segments, such as restating the problem, defining variables, or outlining a general approach. These initial segments can appear highly similar across chains even if their subsequent core reasoning strategies diverge significantly. Consequently, applying similarity-based pruning too early might prematurely eliminate chains that would later develop unique and valuable reasoning paths. To mitigate this, we introduce a *pruning delay*: similarity checks and pruning actions are deferred until chains have progressed beyond a set number of generation steps[1](e.g., `num_steps_to_delay_pruning` analysis intervals, such as 20, as implemented in our system). This delay allows chains to sufficiently develop their distinct approaches before being evaluated for similarity.

3.2 The Pruning Strategy and Algorithm

Once the pruning delay period has passed, our pruning mechanism operates as follows: at each subsequent analysis interval, newly generated *thoughts* (split by stop words like 'Alternatively', 'Wait', 'Another', etc.) are extracted from all active chains. Each new thought from a chain c_i is embedded into a vector representation e_i. This embedding e_i is then compared against the embeddings of previously generated thoughts from all *other* active chains c_j ($j \neq i$) stored in an efficient search index (FAISS) (Douze et al., 2024) using cosine similarity. If the maximum similarity score between e_i and

[1]In our experiments, we define each such step as an interval of 3s. In each step, the system attempts to generate and process thought segments (as defined by stop-word delimiters in Section 4.2) from every currently active reasoning chain.

any thought e_j from a different chain c_j exceeds a predefined threshold τ_{sim}, i.e., $sim(e_i, e_j) > \tau_{sim}$, then chains c_i and c_j are considered a *similar pair*. One of these chains is then selected for pruning. Based on Fig. 2b, which shows that highly similar thoughts (e.g., similarity > 0.9) frequently lead to the same final answer, we select $\tau_{sim} > 0.9$. This high threshold ensures we primarily prune chains that are semantically very close, minimizing the risk of losing valuable, distinct reasoning paths.

Let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of k active reasoning chains. If a new thought from c_i is deemed highly similar to an existing thought in c_j, we employ one of the following strategies to select which chain to prune:

- **Random Pruning ($\mathcal{P}_{rand}$):** Upon identifying a similar pair (c_i, c_j), one chain is chosen uniformly at random to be pruned. This strategy is simple and provides a baseline for comparison. Formally, we prune c_x where x is selected randomly from $\{i, j\}$.
- **Diversity-based Pruning ($\mathcal{P}_{div}$):** This strategy aims to retain the chain that exhibits greater *internal diversity* among its own thoughts. For each chain c_k in the similar pair, we calculate its internal diversity as the mean pairwise cosine similarity of all thought embeddings within that chain, denoted as $S_{internal}(c_k)$.

$$S_{internal}(c_k) = \frac{1}{\binom{m_k}{2}} \sum_{1 \le l < n \le m_k} \text{sim}(e_{k,l}, e_{k,n}) \quad (1)$$

where $\binom{m_k}{2}$ is the number of distinct pairs of thoughts, assuming $m_k \geq 2$. A lower $S_{internal}(c_k)$ indicates greater internal diversity (i.e., thoughts within the chain are less similar to each other). If $S_{internal}(c_i) > S_{internal}(c_j)$, c_i is pruned (as it is less internally diverse).

In all cases, a chain is only pruned if at least one other active chain remains, ensuring the process does not terminate prematurely.

The general procedure for Slim-SC with thought pruning is described in Algorithm 1, with the chain selection logic detailed in Algorithm 2.

4 Experimental Setup

4.1 Datasets and Models

We evaluate on three benchmarks: `GPQA Diamond` (graduate-level STEM questions requiring complex reasoning) (Rein et al., 2023), `AIME-2024` (high-difficulty math problems from a national competition) (AoPS, 2024), and `AQuA-RAT` (algebraic word problems testing multi-step numerical reasoning) (Ling et al., 2017). The models used in our experiments are DeepSeek-R1-Distill-Qwen-14B (R1-Distill) (DeepSeek-AI, 2025b) and Qwen-QwQ-32B (QwQ) (Qwen Team, 2025). For standard Self-Consistency (SC), we first search for the optimal number of chains (N) for each dataset and model. The specific values of N used for each configuration are detailed in Table 7. All pruning baselines and our proposed Slim-SC are evaluated in these optimal N settings.

To measure semantic similarity between thought segments, we first embed all textual outputs using the `all-mpnet-base-v2`[2] Sentence Transformers model (Sentence-Transformers Team, 2021). Subsequently, we compute the cosine similarity between the resulting vector embeddings to obtain pairwise similarity scores.

The embedding and FAISS-based similarity search operations were handled on a separate GPU as model inference to ensure proper isolation, but can be run on the same GPU as model inference. We estimate the computational overhead of these operations to be minimal. The embedding model occupies less than 2% of the GPU memory, and the end-to-end latency from embedding and FAISS search adds an estimated upper-bound delay of only 0.3%. This negligible cost is heavily outweighed by the substantial latency reductions gained from pruning.

4.2 Baseline Methods

We compare Slim-SC against the following established baselines:

1. **Chain-of-Thought (CoT)** (Wei et al., 2023) : This baseline uses standard CoT prompting with greedy decoding, generating a single reasoning path to produce the final answer.

2. **Self-Consistency (SC)** (Wang et al., 2023) : The standard Self-Consistency sampling approach. SC generates N independent reasoning paths and uses majority voting, which is our primary accuracy baseline.

3. **Early-Stopping Self-Consistency (ESC)** (Li et al., 2024) sequentially generates chains in batches (windows) of W chains until it reaches a consensus or until it generates the maximum N chains (followed by majority voting). For a fair

[2] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

comparison, we set N equal to the optimal values of SC and then search for optimal W for ESC. More details are in the Appendix B.1.

4. **Concise CoT with Self-Consistency (CCoT-SC)** (Renze and Guven, 2024) is a naïve baseline adding the instruction "Be concise." to the prompt in the standard SC to shorten reasoning paths. It serves as a straightforward prompting baseline to improve SC efficiency, which mainly depends on the LLMs' instruction-following capability.

5 Results and Discussion

We first showcase the accuracy, resource efficiency, and processing latency advantages of Slim-SC compared to the baseline methods. We then demonstrate the robustness of the approach with Random Pruning (RP) and Diversity-based Pruning (DP) and contrast our informed pruning strategy with naïve alternatives.

5.1 Performance Comparison with Baselines

We summarize all the experimental results for Slim-SC and the baseline methods in Table 1. All reported metrics are averaged over three independent runs to ensure stability.

Slim-SC maintains or improves accuracy over baselines. Across all models and datasets, Slim-SC consistently achieves performance comparable to or better than standard SC. With R1-Distill, Slim-SC (RP) improves accuracy on GPQA Diamond (+0.5 pp over SC) and matches SC on AIME-2024 (82.2%), while outperforming other efficiency-focused baselines like ESC and CCoT-SC. A similar trend holds for QwQ-32B, where Slim-SC variants are highly competitive with SC and notably outperform ESC on complex reasoning tasks (D1 and D2). We attribute Slim-SC's strong performance to its progressive pruning strategy, which, as shown in Appendix A.4, enriches the candidate pool with a higher proportion of correct chains by selectively removing redundant, incorrect ones.

Slim-SC features a superior latency profile. As shown in Table 1, with R1-Distill, Slim-SC (DP) lowers the latency of SC by 45% on GPQA, 43% on AIME, and 11% on AQuA. With QwQ-32B, Slim-SC (DP) trims the latency on GPQA by 20%, on AIME by 8%, and on AQuA by 9%. Slim-SC (RP) achieves comparable gains. Slim-SC provides a clear advantage in inference speed while retaining or improving accuracy.

In contrast, other efficiency-focused baselines struggle with this trade-off. CCoT-SC provides minimal speed-up, while ESC's sequential design creates a severe latency bottleneck. On R1-Distill, ESC is *over 2.9× slower* than Slim-SC (DP) on GPQA. This high latency translates to poor user-facing response times and prolonged GPU occupancy. Furthermore, ESC's overly aggressive early stopping can truncate exploring diverse reasoning pathways prematurely, forming consensus toward incorrect answers, as evidenced by its lower accuracy on R1-Distill D3 (89.5%) compared to the single-chain CoT baseline (89.8%).

Slim-SC effectively reduces token and memory-time cost of SC. By terminating redundant chains early, Slim-SC reduces the total number of generated tokens and mean KV Cache compared to SC. For example, on the complex AIME task with R1-Distill, Slim-SC (DP) slashes token usage by 32% compared to SC (418k vs. 618k) and reduces the mean KV Cache by 10pp compared to SC (46% vs. 56%). A similar trend of substantial token and mean KV Cache reduction is observed across all datasets and models. In many instances Slim-SC is more efficient than CCoT-SC in these two metrics. While Table 1 shows that ESC consistently generates the fewest tokens and has the lowest Mean KV Cache, this raw efficiency comes at a severe price. As analyzed in detail in Appendix A.5, Slim-SC's architecture results in a much lower overall GPU-time cost. By completing tasks significantly faster, it frees up valuable hardware resources far more quickly than ESC's slow, sequential process. ESC's token frugality is a direct consequence of its aggressive early stopping, a mechanism that not only incurs a debilitating latency penalty but also introduces the risk of performance degradation, as previously discussed. Therefore, for practical deployment in high-throughput or time-billed environments, Slim-SC's approach offers a more cost-effective and reliable use of computational resources.

In summary, Slim-SC often matches or surpasses SC in accuracy and outperforms efficiency-focused baselines like ESC and CCoT-SC in most settings. While ESC offers an alternative for extremely token-constrained scenarios, Slim-SC provides a more balanced and practical solution. It is the preferred method when low latency and high accuracy are critical.

5.2 Robustness of Slim-SC

We test how Slim-SC is robust to the choice of pruning rule, pruning delay, and similarity thresh-

Methods	Datasets & Metrics											
	D1	D2	D3	D1	D2	D3	D1	D2	D3	D1	D2	D3
	Accuracy (%)			Latency (s)			Avg. Tokens (thousands)			Mean KVC usage (%)		
R1-Distill												
CoT	58.8±2.3	67.8±9.6	89.8±0.4	112±3	172±18	38±2	7	10	2	2	2	1
SC	63.0±0.6	**82.2**±1.9	90.6±0.4	536±16	942±29	61±3	421±5	618±4	18	47±2	56±2	4
ESC	62.0±0.8	81.1±1.9	89.5±0.5	876±17	1542±100	**51**±1	**262**±3	**392**±22	**5**	**10**	**13**	**1**
CCoT-SC	60.4±1.5	80.0±3.3	89.9±0.2	505±32	797±64	58±2	402±6	572±7	17	46±3	44±15	3
Slim-SC (DP)	62.5±1.1	**82.2**±1.9	**90.8**±0.2	**296**±84	**536**±167	54±2	278±67	418±107	16±1	35±9	46±12	3
Slim-SC (RP)	**63.5**±1.8	**82.2**±1.9	90.2	381±126	664±139	56±1	328±77	486±96	17	43	49±8	3
QwQ-32B												
CoT	64.4±3.7	76.7±3.3	90.6±0.4	157±12	300±41	59±1	8	13±1	3	2	3	1
SC	**66.8**±1.3	83.3	91.5±0.5	312±28	479±75	92±1	119±1	105	24±1	16	15±1	4
ESC	66.0±1.1	80.0	**91.6**±0.2	315±20	574±17	91±4	**26**±2	**47**±2	**8**	**3**	**5**	**1**
CCoT-SC	66.7±1.8	**84.4**±1.9	91.5±0.2	286±27	435±65	89±5	111	99±2	22	15	15	3
Slim-SC (DP)	64.6±1.5	**84.4**±1.9	91.5±0.2	**250**±9	442±19	**84**	95±6	104±2	20±1	12±1	15	3
Slim-SC (RP)	65.8±1.3	82.2±1.9	**91.6**±0.6	276±15	**431**±17	**84**±2	109±11	97±6	20±2	15±1	14±1	3

Table 1: Accuracy, mean processing latency per question in seconds, mean number of completion tokens per question, and mean Key-Value Cache usage per question for baselines and Slim-SC on three datasets.
Dataset legend: GPQA Diamond as D1, AIME-2024 as D2, AQuA-RAT as D3.
Result highlights: Slim-SC achieves the same or higher accuracy than all the baselines, except vs. SC with QwQ-32B for GPQA. Slim-SC also delivers much lower latency for GPQA and AIME. Slim-SC uses fewer tokens and memory compared to all methods, except ESC.

olds. The threshold values of Slim-SC for Table 1 are presented in Table 8 and Table 9.

5.2.1 Similarity Pruning Algorithms

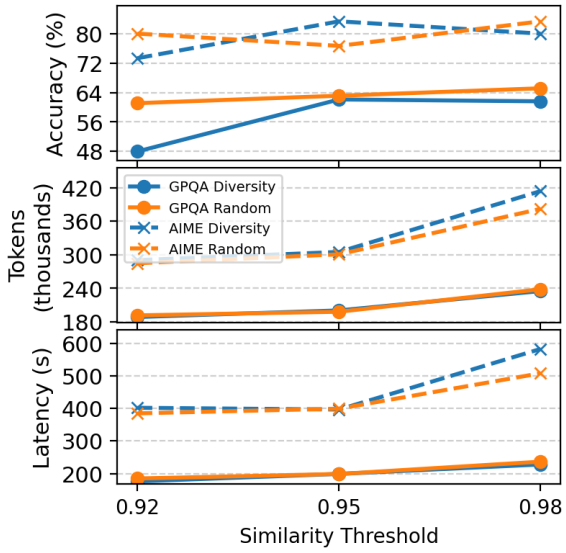


Figure 3: Accuracy, mean completion tokens per question, and processing latency of Slim-SC on R1-Distill when varying the similarity threshold for the diversity and random-based pruning methods in Slim-SC.

Random Selection is simple and effective while Internal Diversity delivers token efficiency. Our main experiments compare two selection heuristics for pruning a similar pair: Random Pruning (RP) and Diversity-based Pruning (DP). As shown in Table 1, Slim-SC (RP) consistently delivers strong and stable accuracy, making it a simple, effective, and robust default strategy. Slim-SC (DP) offers an alternative trade-off; it often yields lower token usage by operating at a slightly lower similarity threshold (see Fig. 3), which can be beneficial in resource-constrained settings, sometimes at the cost of a minor drop in accuracy.

5.2.2 Ablation Study

To validate that Slim-SC's performance stems from its informed, similarity-based strategy, we compare it against two naïve pruning baselines:

- **Global Random Pruning**: At each analysis step, one active chain is randomly selected from the entire pool and pruned, without considering semantic content. This baseline tests whether deliberate pruning is necessary.
- **Least Similar Pruning**: At each step, the pair of thoughts from different chains with the lowest cosine similarity is identified. One of these two chains is then randomly pruned, actively removing diversity. This baseline tests whether pruning similar chains is indeed beneficial.

As shown in Table 2, both strategies lead to a significant degradation in accuracy compared to Slim-SC. Least Similar Pruning is particularly harmful, with its accuracy on `AIME-2024` dropping to 74.4%-a nearly 8pp loss compared to Slim-SC. This confirms that reasoning diversity is crucial for SC and

Methods	Datasets		
	D1	D2	D3
	Accuracy (%)		
CoT	64.4±3.7	76.7±3.3	90.6±0.4
SC	**66.8**±1.3	83.3	91.5±0.5
GlobalRP	64.1±1.3	77.8±1.9	91.2±0.6
LeastSimP	62.6±3.0	74.4±8.3	91.1±0.2
Slim-SC (DP)	64.6±1.5	**84.4**±1.9	91.5±0.2
Slim-SC (RP)	65.8±1.3	82.2±1.9	**91.6**±0.6

Table 2: Compared to Slim-SC, accuracy drastically degrades when naïve pruning strategies like Global Random Pruning (GlobalRP) and Least Similar Pruning (LeastSimP) are used, when running the three datasets on QwQ-32B.
Dataset legend: GPQA Diamond as D1, AIME-2024 as D2, AQuA-RAT as D3.

that actively destroying it is detrimental. Global Random Pruning, while less harmful, is still clearly suboptimal as its unguided nature risks removing valuable chains. This ablation provides strong evidence that Slim-SC's performance gains are directly attributable to its core principle: intelligently targeting redundancy while preserving diversity.

5.3 Hyperparameter Sensitivity Analysis

5.3.1 The Similarity Threshold

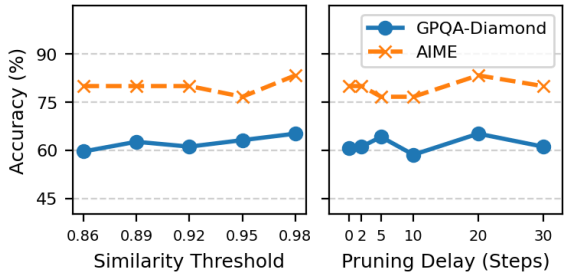


Figure 4: Accuracy remains stable for wide ranges of similarity thresholds and pruning delays when tested with R1-Distill. We choose the threshold of 0.98 and delay of 20 steps as more optimal.

Performance is stable and robust at high similarity thresholds.

As shown in Fig. 4 (left), accuracy remains high and stable for $\tau_{sim} \geq 0.95$, typically peaking at $\tau_{sim} = 0.98$. This empirical finding is not arbitrary; it is directly explained by the underlying similarity distributions of correct and incorrect chains, as detailed in Section 2.4 and visualized in the appendix (Figure 7).

An analysis of these distribution plots reveals a crucial pattern. While incorrect chains are consistently more self-similar than correct ones across all thresholds, this disparity is often most pronounced around the $\tau_{sim} = 0.98$ mark. For instance, on `AIME-2024` with R1-Distill (Fig. 2b), the proportion of incorrect-incorrect pairs among all similar pairs is maximized near this threshold, while the proportion of correct-correct pairs is minimized. This indicates that $\tau_{sim} = 0.98$ is often the *sweet spot* where our pruning strategy is maximally effective at its intended goal: disproportionately removing redundant incorrect chains while preserving the diversity among correct ones. This cleansing of the candidate pool provides a strong mechanistic justification for the peak in final accuracy observed at this threshold.

This insight leads to our practical recommendation: a high, fixed threshold like *0.98 is not only a safe and robust choice but is also grounded in the observed clustering behavior of reasoning chains*. This allows practitioners to achieve optimal or near-optimal performance without needing to perform per-dataset tuning for this hyperparameter.

5.3.2 Pruning Delay

Accuracy is stable across delay steps, peaking at a moderate delay of 20 steps. With each step defined as an interval of 3s, Fig. 4 (right) demonstrates the benefit of a pruning delay. While accuracy is relatively stable across various delay durations, it consistently peaks when the delay is set to 20 analysis steps for both `GPQA-Diamond` (63.5%) and `AIME-2024` (82.2%). This indicates that allowing chains a moderate period of 20 steps to develop their initial reasoning and diverge from one another is optimal before the similarity checks and pruning commence. This delay effectively preserves potentially correct and unique reasoning paths that might otherwise be pruned too early.

6 Related Work

6.1 Test-Time Scaling (TTS)

TTS refers to methods that enhance reasoning performance by investing additional computation at inference time, without modifying the model's parameters (Zhang et al., 2025b). Existing TTS approaches can be categorized into sequential and parallel scaling. Sequential scaling extends the reasoning process by generating a series of intermediate steps before providing the final answer, as represented by Chain-of-Thought (CoT) prompting

(Wei et al., 2023). The intermediate outputs can be further refined using methods like Self-Refine (Madaan et al., 2023) or s1 (Muennighoff et al., 2025). However, the single-chain problem-solving trajectory can be compromised by error propagation, i.e., mistakes made in early steps may mislead subsequent reasoning.

Parallel scaling, by contrast, explores diverse reasoning paths simultaneously and selects the most plausible final answer based on certain criteria. Typical approaches include Best-of-N (BoN) sampling (Lightman et al., 2024) and SC decoding (Wang et al., 2023). Both methods generate solutions in parallel, but differ in selection strategy: BoN uses a pretrained reward model to score the solutions based on its likelihood of yielding the correct answer (Cobbe et al., 2021), whereas SC relies on majority voting among the predicted answers. Recent research explores latent reasoning with SC (Xu et al., 2025b), further amplifying the benefit of both explicit and implicit reasoning (Xu et al., 2025a) at test time. In summary, parallel scaling increases the chances of finding a correct solution and circumvents the potential error aggregation in sequential scaling. While it often enhances complex reasoning tasks, the effectiveness of TTS often requires non-trivial computational cost.

Besides TTS, system support that explores elastic inference scaling that right-sizes GPU cluster resources can improve resource efficiency while maintaining the accuracy (Fu et al., 2024; Zhong et al., 2024; Zhang et al., 2025a).

6.2 Pruning Methods for Parallel Scaling

Parallel scaling typically requires generating N full reasoning paths, each leading to a final answer. When N is large enough, the performance gain can match that of post-training approaches (Sun et al., 2024). However, generating all the full paths is computationally expensive, and many paths contribute little to the final decision. To address this, recent works focus on: 1) dynamically reducing N; and 2) halting low-quality paths early.

Instead of fixing the number of paths N for all questions, methods such as Adaptive-Consistency (AC) (Aggarwal et al., 2023), Early-Stopping Self-Consistency (ESC) (Li et al., 2024), Difficulty-Adaptive Self-Consistency (DSC) (Wang et al., 2025a), and Reasoning-Aware Self-Consistency (RASC) (Wan et al., 2025) incrementally generate reasoning paths in small batches until the answers reach a consensus or a cap of N is met. Specifically, ESC theoretically proves that this incremental sampling strategy maintains accuracy to a high degree. DSC measures question difficulty using an LLM to degenerate SC to single-chain CoT ($N = 1$) for easy questions. RASC introduced a nuanced evaluation by incorporating both the answer consistency and reasoning quality via a weighted voting (similar to Taubenfeld et al., 2025). Additionally, to accelerate finding the confident answer, Path-Consistency (Zhu et al., 2025) reuses early reasoning steps from a confident path to guide the generation of subsequent paths. *While effective, all these methods introduce latency due to sequential sampling, trading time for memory efficiency.*

Other approaches fix N but halt the generation of unpromising paths early based on certain metrics. Fast BoN (Sun et al., 2024) uses the confidence scores from a reward model, whereas Self-Truncation BoN (Wang et al., 2025b) introduces an internal consistency metric that computes the average squared distance between a path and all others. It keeps generating for *a fixed time window* c and then prunes low-consistency paths. A similar strategy has also been recently applied in SC, i.e., Reasoning Pruning Perplexity Consistency (RPC) (Zhou et al., 2025), which uses the model's own confidence to generate the correct answer, $p(y|x)$, after generating a reasoning step x. This paper contributes to this line by proposing a new efficient self-consistency method, Slim-SC, where we simply prune the chains based on the similarity of thoughts across the chains.

Conclusion

This work introduces Slim-SC, a step-wise thought pruning method that improves the efficiency of SC without sacrificing accuracy. Motivated by the observation that correct and incorrect chains form distinct clusters, Slim-SC prunes highly similar chains to remove redundancy while preserving the diversity of reasoning paths. Experiments show that Slim-SC substantially lowers latency and KVC usage by up to 45% and 26%, respectively, while offering a more robust accuracy-latency trade-off than methods like ESC. Moreover, Slim-SC's similarity-based pruning outperforms naïve pruning, confirming that preserving reasoning diversity is key to complex problem-solving. Finally, its simple Random Pruning (RP) variant makes Slim-SC a practical tool for deploying LLMs in latency-sensitive applications.

Limitations

While Slim-SC demonstrates promise for improving SC-based efficiency, several limitations remain:
Metric Design: Slim-SC uses similarity patterns to surface correct chains, but a more effective and lightweight metric is needed to fully bridge the gap between oracle and actual accuracy (Fig. 2a).
Pruning vs. Reuse: Our method prunes similar chains outright. Future work could merge partial reasoning segments or KV-cache states to improve both accuracy and computational reuse.
Scalability: Evaluation should be extended to larger models and more diverse datasets to assess robustness and generalizability, as our study is currently limited to math reasoning with R1-Distill-14B and QwQ-32B.

Acknowledgments

The authors thank the members of the HyScale lab at NTU Singapore for their constructive discussions and feedback on this work, and Vlad Pandelea for the guidance provided during the early stages of this work. This project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (Project RS26/23). Xu Guo is supported by the Wallenberg-NTU Presidential Postdoctoral Fellowship and Colin Hong is supported by the A*STAR Graduate Scholarship.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. 2023. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12375–12396, Singapore. Association for Computational Linguistics.

AoPS. 2024. Aime problems and solutions.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *Preprint*, arXiv:2407.21787.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

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

DeepSeek-AI. 2025a. Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

DeepSeek-AI. 2025b. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

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

Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. Serverlessllm: Low-latency serverless inference for large language models. *Preprint*, arXiv:2401.14351.

Xu Guo. 2025. Measuring reasoning utility in llms via conditional entropy reduction. *arXiv preprint arXiv:2508.20395*.

Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. 2025. Don't overthink it. preferring shorter thinking chains for improved llm reasoning. *Preprint*, arXiv:2505.17813.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2024. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *The Twelfth International Conference on Learning Representations*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *ACL*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

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. *Preprint*, arXiv:2501.19393.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Qwen Team. 2025. Qwen-qwq-32b.

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

Matthew Renze and Erhan Guven. 2024. The benefits of a concise chain of thought on problem-solving in large language models. *Preprint*, arXiv:2401.05618.

Sentence-Transformers Team. 2021. all-mpnet-base-v2.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. 2025. Confidence improves self-consistency in llms. *Preprint*, arXiv:2502.06233.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. 2025. Reasoning aware self-consistency: Leveraging reasoning paths for efficient llm sampling. *Preprint*, arXiv:2408.17017.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025a. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6904–6917, Albuquerque, New Mexico. Association for Computational Linguistics.

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

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2025b. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. *Preprint*, arXiv:2503.01422.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

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

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025b. Softcot++: Test-time scaling with soft chain-of-thought reasoning. *Preprint*, arXiv:2505.11484.

Dingyan Zhang, Haotian Wang, Yang Liu, Xingda Wei, Yizhou Shan, Rong Chen, and Haibo Chen. 2025a. Blitzscale: Fast and live large model autoscaling with o(1) host caching. *Preprint*, arXiv:2412.17246.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Zhihan Guo, Yufei Wang, Niklas Muennighoff, Irwin King, Xue Liu, and Chen Ma. 2025b. What, how, where, and how well? a survey on test-time scaling in large language models. *Preprint*, arXiv:2503.24235.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. *Preprint*, arXiv:2401.09670.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

Zhi Zhou, Tan Yuhao, Zenan Li, Yuan Yao, Lan-Zhe Guo, Xiaoxing Ma, and Yu-Feng Li. 2025. Bridging internal probability and self-consistency for effective and efficient llm reasoning. *Preprint*, arXiv:2502.00511.

Jiace Zhu, Yingtao Shen, Jie Zhao, and An Zou. 2025. Path-consistency: Prefix enhancement for efficient inference in llm. *Preprint*, arXiv:2409.01281.

A Analysis of Self-Consistency

A.1 Divergence in the Mean Completion Tokens

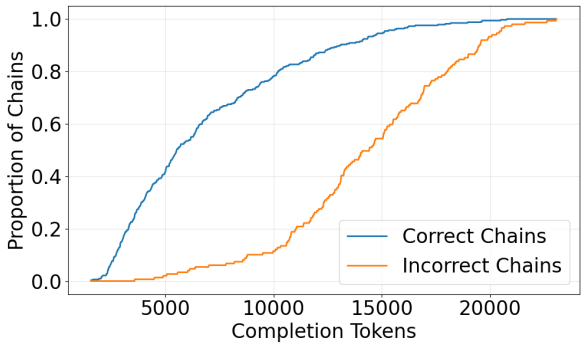


Figure 5: CDF of chains by completion tokens using R1-Distill on AIME with N=16

Our analysis in Section 2.3 highlights that standard SC often waits for long, incorrect chains. This observation is further substantiated by examining the distribution of completion tokens for correct versus incorrect chains. Fig. 5 presents the Cumulative Distribution Function (CDF) of completion tokens for chains generated by R1-Distill on the `AIME` dataset (with $N = 16$).

The CDF reveals a distinct divergence:

- **Correct chains tend to be shorter.** A significant proportion of correct chains complete with fewer tokens. For instance, approximately 60% of correct chains are completed within 7,500 tokens.
- **Incorrect chains are generally longer.** The CDF for incorrect chains rises much more slowly, indicating that a larger number of tokens is typically required before an incorrect chain completes. Only about 20% of incorrect chains are completed within 10,000 tokens, whereas nearly 80% of correct chains have finished by this point.

This divergence in completion token lengths, where incorrect chains are frequently longer, reinforces the inefficiency of the *wait-for-all* SC strategy. It also supports the rationale behind pruning, as selectively removing chains (especially those that are both incorrect and lengthy, or highly similar to other incorrect, lengthy chains as suggested by Fig. 2b) could save substantial computational resources without necessarily compromising the chances of finding a correct answer among the shorter, more efficient correct chains. This observed difference in token usage profiles further motivates strategies like Slim-SC that aim to curtail the generation of overly long or redundant paths.

A.2 *Wait-for-all* effect in SC

Fig. 6 demonstrates that the *wait-for-all* problem is not only experienced at $N = 64$ as shown in Fig. 1c, but experienced to an increasing degree as N increases.

A.3 Categorizing similar chains

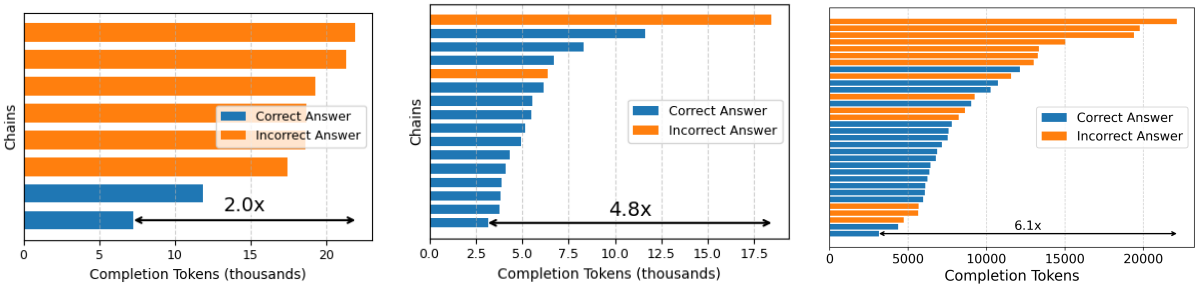
To produce the results in Fig. 2b, we begin with the textual outputs from our SC ($N = 64$) experiment on R1-Distill using the GPQA Diamond dataset. We then apply a modified version of Algorithm 1. Specifically, we define a step as 100 tokens, based on our experimental estimate of the average number of tokens generated per step in the final setup. We set $D_{\text{steps}} = 20$ and proceed with the algorithm, except that instead of performing the pruning steps (lines 19–23), we record all chain pairs that were deemed similar and classify each pair based on whether both chains were correct, both incorrect, or one correct and one incorrect. We include similar charts for the rest of the models and datasets at their corresponding ideal N (from Table 7) in Figure 7.

A.4 Impact of Pruning on the Quality of the Candidate Pool

Our core motivation, as discussed in Section 2.4, is that incorrect reasoning chains tend to form denser semantic clusters than correct ones. While Fig. 2b illustrates this by showing the composition of similar pairs, it does not directly demonstrate that our pruning method improves the quality of the final candidate set used for voting.

To provide this direct evidence, we analyzed the average proportion of correct chains remaining in the candidate pool after pruning, compared to the initial proportion in standard SC. The results are presented in Table 3 and Table 4.

The results in these tables confirm our hypothesis. Across both models, **Slim-SC (RP) increases**

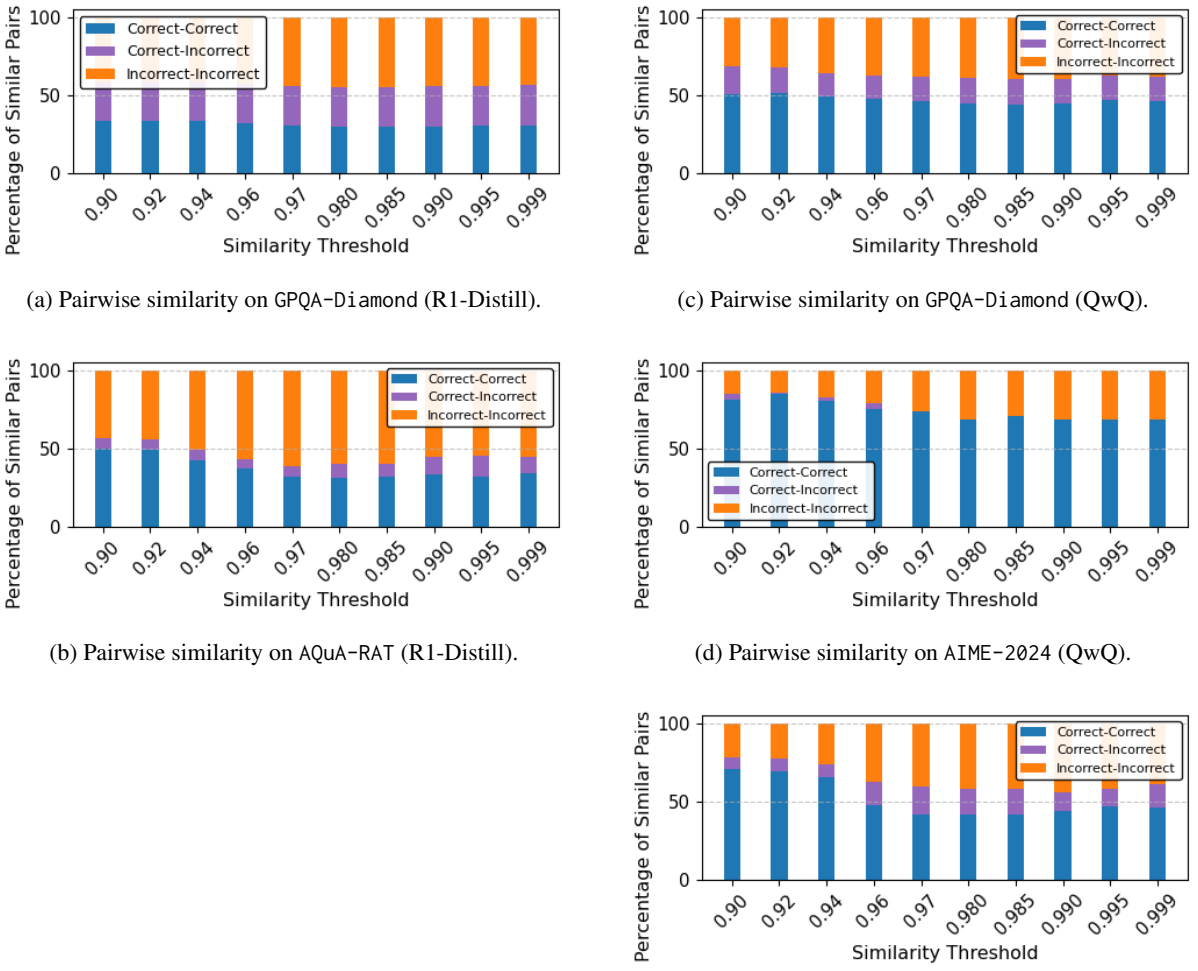


(a) SC ($N = 8$) for R1-Distill on AIME-2024 Q27.

(b) SC ($N = 16$) for R1-Distill on AIME-2024 Q29.

(c) SC ($N = 32$) for R1-Distill on AIME-2024 Q29.

Figure 6: SC *wait-for-all* effect at other Ns.



(a) Pairwise similarity on `GPQA-Diamond` (R1-Distill).

(c) Pairwise similarity on `GPQA-Diamond` (QwQ).

(b) Pairwise similarity on `AQuA-RAT` (R1-Distill).

(d) Pairwise similarity on `AIME-2024` (QwQ).

(e) Pairwise similarity on `AQuA-RAT` (QwQ).

Figure 7: Pairwise similarities using R1-Distill (left) and QwQ (right) across various datasets.

the proportion of correct chains in 5 out of 6 experimental settings compared to standard SC. This provides strong, direct evidence that our similarity-based pruning strategy is working as intended: it preferentially removes redundant incorrect chains more often than correct ones. This *cleansing* of the candidate pool mechanistically explains why Slim-SC can maintain or even improve upon the final accuracy of SC, despite using fewer computational resources.

Method	D1	D2	D3
SC	0.578	0.717	0.879
Slim-SC (DP)	0.585	0.716	**0.905**
Slim-SC (RP)	**0.599**	**0.735**	0.900

Table 3: R1-Distill-Qwen-14B: Average proportion of correct chains in the final candidate pool.

Method	D1	D2	D3
SC	0.637	0.775	0.916
Slim-SC (DP)	0.626	0.767	0.915
Slim-SC (RP)	**0.657**	0.773	**0.917**

Table 4: QwQ-32B: Average proportion of correct chains in the final candidate pool.

A.5 Understanding KV Cache Efficiency: Peak vs. Duration

While Table 5 reports Mean Peak KV Cache usage, this single metric can be misleading if not considered in the context of processing time. A more holistic measure of resource cost is the *GPU-time product*, which reflects the total duration a hardware accelerator is occupied. For time-billed cloud instances or high-throughput on-premise clusters, minimizing the total time a GPU is reserved for a single request is paramount for both cost-efficiency and serving capacity.

As shown in Table 5, ESC achieves the lowest Mean Peak KV Cache usage. However, its sequential nature leads to prolonged GPU occupancy, as it processes small batches over a much longer duration (see Latency in Table 1). This results in a high overall GPU-time cost.

In contrast, Fig. 8 illustrates the superior resource-time profile of Slim-SC.

- **Initial Peak:** Slim-SC launches all chains in parallel, causing its KV Cache usage to momentarily peak at a level similar to standard SC. This initial ramp-up explains the high *Mean Peak* values in the table.
- **Rapid Decline:** Critically, once the pruning mechanism activates, the KV cache usage for Slim-SC (orange line) rapidly plummets. The resources are freed up much faster as redundant chains are terminated.
- **Sustained Cost of Baselines:** Standard SC (blue line) exhibits sustained high KV Cache usage for a much longer period until chains naturally complete. ESC, while not plotted, would show a sawtooth pattern of smaller peaks over an even longer timeframe, occupying the GPU for the entire extended latency period.

Methods	Datasets		
	D1	**D2**	**D3**
	Mean Peak KVCache (%)		
R1-Distill			
CoT	3	5	1
SC	86	$94_{\pm 2}$	$8_{\pm 2}$
ESC	21	$28_{\pm 1}$	2
CCoT-SC	84	$81_{\pm 15}$	6
Slim-SC (DP)	82	$92_{\pm 2}$	6
Slim-SC (RP)	85	93	6
QwQ-32B			
CoT	3	5	1
SC	31	$30_{\pm 1}$	7
ESC	5	9	2
CCoT-SC	29	28	6
Slim-SC (DP)	$25_{\pm 2}$	$29_{\pm 1}$	6
Slim-SC (RP)	$29_{\pm 2}$	$27_{\pm 2}$	6

Table 5: Mean Peak Key-Value Cache usage per question for baselines and Slim-SC on three datasets. **Dataset legend**: GPQA Diamond as D1, AIME 2024 as D2, AQuA-RAT as D3.

Slim-SC's strategy of utilizing higher memory for a short burst is significantly more efficient in terms of total GPU-time cost than ESC's strategy of using low memory over a prolonged period. By finishing tasks faster, Slim-SC frees up valuable GPU resources for subsequent requests, making it a more practical and cost-effective solution for real-world, latency-sensitive applications.

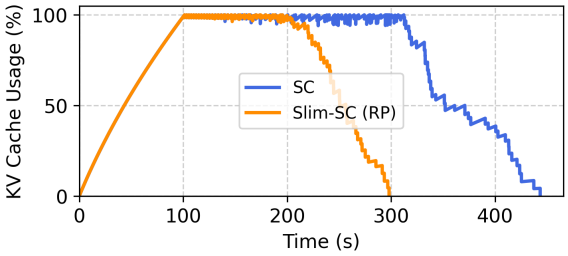
B Experimental Details

B.1 ESC Reproduction Details.

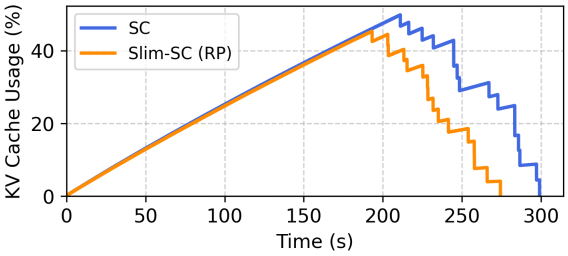
Following the original work's setup for $N_{max} = 40$ (where they used $W = 5$, i.e., 8 windows), we aim for a similar number of maximum windows. Thus, we set $W = \max(2, \lceil N/8 \rceil)$. The $\max(2, \dots)$ ensures that $W \geq 2$ to avoid degenerating to simple CoT when N is small. For instance, if $N = 64$, $W = 8$. If $N = 8$, $W = 2$. The optimal setting searched for ESC is presented in Table 6.

B.2 Prompt Templates

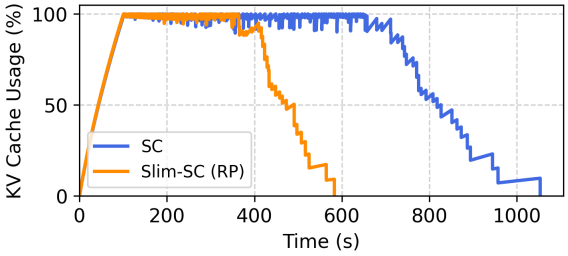
To ensure standardization, the following are the templates for the prompts we used for the



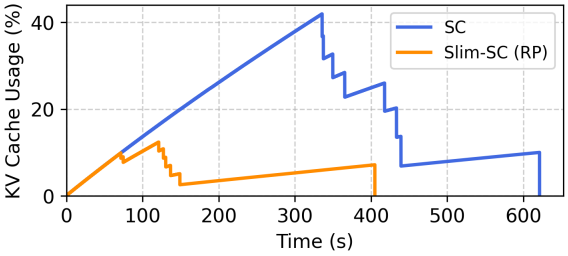
(a) GPQA-Diamond Q2 (R1-Distill).



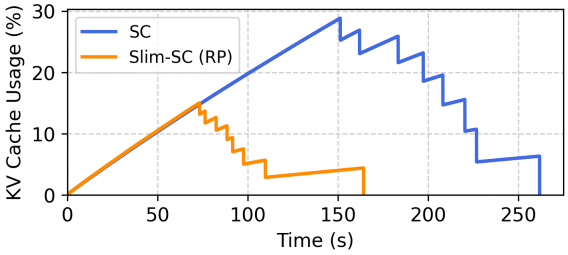
(d) GPQA-Diamond Q70 (QwQ).



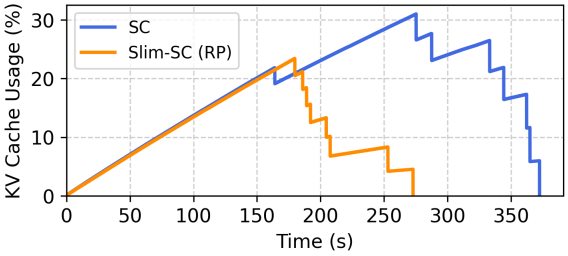
(b) AIME-2024 Q22 (R1-Distill).



(e) AIME-2024 Q8 (QwQ).



(c) AQuA-RAT Q70 (R1-Distill).



(f) AQuA-RAT Q242 (QwQ).

Figure 8: Comparison of KVCache usage (%) over time between SC and Slim-SC (RP) for R1-Distill (left) and QwQ (right).

Model	GPQA D.	AIME'24	AQuA-R.
R1-Distill	8	8	2
QwQ-32B	2	2	2

Table 6: The window size parameter searched for ESC.

CoT/SC (AIME-2024)

Prompt:
Answer the following math problem.
The last line of your response should be your integer answer within boxed{{}}.
{question_text}
Put your final answer within boxed{{}}
Think step by step before answering.

`GPQA-Diamond`, `AIME-2024` and `AQuA-RAT` benchmarks.

CCoT (AIME-2024)

Prompt:
Answer the following math problem.
The last line of your response should be your integer answer within boxed{{}}.
{question_text}
Put your final answer within boxed{{}}
Think step by step before answering. Be concise.

CoT/SC (AQuA-RAT)

Prompt:
Answer the following multiple-choice question. Think step-by-step to reach the solution. Conclude your response with a single line containing the answer letter formatted exactly as 'Answer: $LETTER'.
Question: {question_text}
Options: {options}

CCoT (AQuA-RAT)

Prompt:
Answer the following multiple-choice question. Think step-by-step to reach the solution. Be concise. Conclude your response with a single line containing the answer letter formatted exactly as 'Answer: $LETTER'.
Question: {question_text}
Options: {options}

CoT/SC (GPQA-Diamond)

Prompt:
Answer the following multiple-choice science question. Think step-by-step to reach the solution. Conclude your response with a single line containing the answer letter formatted exactly as 'Answer: $LETTER'.
Question: {question_text}
Options: {options}

CCoT (GPQA-Diamond)

Prompt:
Answer the following multiple-choice science question. Think step-by-step to reach the solution. Be concise. Conclude your response with a single line containing the answer letter formatted exactly as 'Answer: $LETTER'.
Question: {question_text}
Options: {options}

B.3 Optimal N for standard SC

Model	GPQA D.	AIME'24	AQuA-R.
R1-Distill	64	64	8
QwQ-32B	16	8	8

Table 7: The optimal N searched for the baseline Self-Consistency on each dataset and model.

B.4 Optimal thresholds for Slim-SC

Model	GPQA D.	AIME'24	AQuA-R.
R1-Distill	0.95	0.95	0.95
QwQ-32B	0.95	0.98	0.98

Table 8: The similarity threshold empirically chosen for Slim-SC (DP).

Model	GPQA D.	AIME'24	AQuA-R.
R1-Distill	0.98	0.98	0.98
QwQ-32B	0.98	0.98	0.98

Table 9: The similarity threshold empirically chosen for Slim-SC (RP).

Algorithm 1 Slim-SC with Similarity Pruning

Input: Question Q, Initial number of chains N, Similarity threshold τ_{sim}, Pruning strategy $\mathcal{P}_{strategy} \in \{\mathcal{P}_{rand}, \mathcal{P}_{div}\}$, Pruning delay parameters $D_{thoughts}, D_{steps}$.
Output: Final voted answer A_{final}.
Initialize N chains $C = \{c_1, \ldots, c_N\}$ generating thoughts for Q.
Initialize empty thought embedding index F_{idx}.
Initialize empty set of completed thoughts $T_{all} = \emptyset$.
$current\_step \leftarrow 0$.
while any chain in C is active AND $|C| > 1$ **do**
    $current\_step \leftarrow current\_step + 1$.
    Wait for new thought segments from active chains or analysis interval.
    for each active chain $c_i \in C$ **do**
        Extract newly generated thought th_{new} from c_i.
        if th_{new} is not null **then**
            $e_{new} \leftarrow$ Embed(th_{new}).
            Add e_{new} to c_i's internal list of thought embeddings.
            $num\_thoughts_i \leftarrow$ count of thoughts in c_i.
            if $current\_step > D_{steps}$ **then** ▷ Pruning delay check
                $(e_{neighbor}, c_{neighbor}) \leftarrow$ FindNearestNeighbor$(F_{idx}, e_{new}, \text{exclude_chain} = c_i)$.
                if $c_{neighbor}$ is not null AND Similarity$(e_{new}, e_{neighbor}) > \tau_{sim}$ **then**
                    $c_{prune} \leftarrow$ SelectChainToPrune$(c_i, c_{neighbor}, \mathcal{P}_{strategy})$ (Algorithm 2).
                    if $|C| - |\{\text{chains marked for pruning}\}| > 1$ **then**
                        Mark c_{prune} as inactive.
                        Stop generation for c_{prune}.
                        Remove embeddings of c_{prune} from F_{idx}.
                        continue to next chain or step.
                    end if
                end if
            end if
            Add $(e_{new}, c_i, num\_thoughts_i)$ to F_{idx}.
        end if
        if c_i completes generation or reasoning phase ends for c_i **then**
            Mark c_i so it's no longer checked for pruning but continues if not pruned.
        end if
    end for
    if all remaining active chains have completed reasoning **then**
        break ▷ No more pruning possible
    end if
end while
Wait for all remaining active chains in C to complete generation.
Collect final answers from non-pruned, completed chains.
$A_{final} \leftarrow$ MajorityVote(collected answers).
return A_{final}.

Algorithm 2 SelectChainToPrune($c_a, c_b, \mathcal{P}_{strategy}$)

Input: Chains c_a, c_b; Pruning strategy $\mathcal{P}_{strategy}$.
Output: Chain selected for pruning.
if $\mathcal{P}_{strategy} = \mathcal{P}_{rand}$ **then**
    return Randomly chosen chain from $\{c_a, c_b\}$.
else if $\mathcal{P}_{strategy} = \mathcal{P}_{div}$ **then**
    $S_{internal}(c_a) \leftarrow$ CalculateInternalDiversityScore(c_a).
    $S_{internal}(c_b) \leftarrow$ CalculateInternalDiversityScore(c_b).
    if $S_{internal}(c_a) > S_{internal}(c_b)$ **then** ▷ Higher score means less diverse
        return c_a.
    else if $S_{internal}(c_b) > S_{internal}(c_a)$ **then**
        return c_b.
    else ▷ Tie-breaking: internal diversity scores are equal.
        $num\_thoughts_a \leftarrow$ count of thoughts in c_a.
        $num\_thoughts_b \leftarrow$ count of thoughts in c_b.
        if $num\_thoughts_a \leq num\_thoughts_b$ **then** ▷ Heuristic: prune chain w/ fewer/eq thoughts.
            return c_a.
        else
            return c_b.
        end if
    end if
end if
procedure CALCULATEINTERNALDIVERSITYSCORE(c)
    $E_c \leftarrow$ list of thought embeddings for chain c.
    if $|E_c| < 2$ **then return** 0.0.
    end if
    $sum\_sim \leftarrow 0$; $pair\_count \leftarrow 0$.
    for $i \leftarrow 0$ to $|E_c| - 2$ **do**
        for $j \leftarrow i + 1$ to $|E_c| - 1$ **do**
            $sum\_sim \leftarrow sum\_sim +$ Similarity$(E_c[i], E_c[j])$.
            $pair\_count \leftarrow pair\_count + 1$.
        end for
    end for
    return $sum\_sim/pair\_count$ if $pair\_count > 0$ else 0.0.
end procedure