TOKENDISTILL: COMPRESSING COT VIA ITERATIVE TOKEN DISTILLATION WITH CONTINUOUS PROMPT TRANSFORMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-Thought (CoT) prompting can substantially improve the final-answer accuracy of large language models (LLMs) by enabling step-by-step reasoning. However, the intermediate tokens it generates incur additional computational and memory overhead, thereby increasing deployment costs. Existing CoT compression methods either attain limited compression ratios, rely on auxiliary external models, or require extra comparison samples during optimization, which introduces considerable training overhead. To address these limitations, we propose TokenDistill, a novel training pipeline that enables the model itself to iteratively distill CoT tokens in an adaptive manner, achieving aggressive CoT compression while largely preserving final-answer accuracy. At the core of TokenDistill is an iterative prompt transformation mechanism, which enables the model to internalize the requirement for concise outputs from explicit prompting into its own generation behavior. Specifically, we first augment the original input prompts with a concise-output instruction to obtain concise outputs. We then fine-tune the model using the original inputs without the concise-output instruction and the generated concise outputs as supervised input-output pairs, thereby internalizing the concise-output behavior into the model. Furthermore, we repeat the aforementioned procedure over multiple rounds, enabling the model to progressively push its compression limits beyond the previously attained level. Extensive experiments demonstrate that TokenDistill consistently outperforms state-of-the-art methods in CoT compression, incurring only minimal accuracy loss and in some cases even slight accuracy gains.

1 INTRODUCTION

With the continuous evolution of language models (Hochreiter & Schmidhuber, 1997; Vaswani et al., 2017; Brown et al., 2020), researchers have increasingly explored their applications beyond daily conversation (Adiwardana et al., 2020; Zhang et al., 2020; Roller et al., 2021), extending them to mathematical, scientific, and engineering domains (Taylor et al., 2022; Gao et al., 2023; Rozière et al., 2023; Lightman et al., 2024). However, solving problems in these domains often requires models to be able to perform complex logical reasoning. Directly deriving the final answer from the input in a single step is neither practical nor reliable. To overcome this limitation, Wei et al. (2022) introduced chain-of-thought (CoT) prompting to guide large language models (LLMs) to generate intermediate reasoning steps. Subsequently, extensive studies and experiments (Kojima et al., 2022; Zhang et al., 2023; Yao et al., 2023) have shown that CoT can help LLMs decompose complex problems into a series of manageable subtasks, thereby improving problem-solving performance.

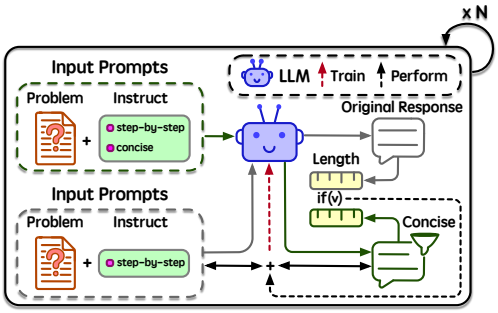

Figure 1: Overview of TokenDistill.

Although CoT has proven effective, the intermediate reasoning steps it generates substantially increase the total number of output tokens (Chiang & Lee, 2024). Since mainstream LLM architectures rely on autoregressive decoding (Brown et al., 2020) with attention (Vaswani et al., 2017), each newly

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

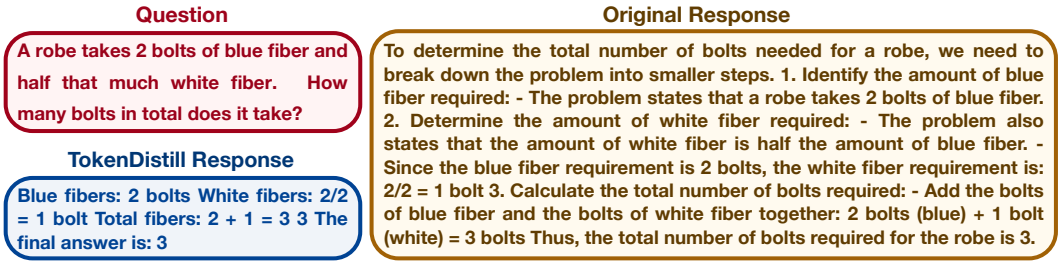

Figure 2: Example of TokenDistill compression. TokenDistill enables the model to distill the essential reasoning steps, thereby significantly compressing the output while preserving answer correctness.

generated token must attend to all previous tokens. Consequently, the additional intermediate reasoning steps introduced by CoT incur substantial overhead, as attention-related computation scales approximately quadratically with sequence length. Additionally, each newly generated token introduces additional cached states that must be retained for subsequent decoding steps, leading to roughly linear growth in memory usage.

To mitigate the excessive resource overhead, several studies have sought to compress CoT. Prompt-based methods incorporate conciseness-related instructions to elicit shorter responses (Renze & Guven, 2024; Nayab et al., 2024; Lee et al., 2025). While effective, such methods generally achieve only limited compression, as they do not update the model. To attain stronger compression, recent studies resort to model training, either by introducing length-aware rewards (Team et al., 2025; Aggarwal & Welleck, 2025; Zhang et al., 2025) or by constructing training data with auxiliary models (Pan et al., 2024; Kang et al., 2025; Xia et al., 2025). However, these methods often require additional comparison samples or external model assistance, thereby incurring considerable training and implementation overhead. Moreover, despite generally outperforming prompt-based methods, they still achieve only limited compression ratios.

To address these challenges, we propose TokenDistill, a novel training pipeline that enables the model to adaptively and iteratively distill its own CoT tokens without any additional comparison samples or external model assistance. At the core of TokenDistill is a continuous prompt transformation mechanism that repeatedly adds and removes a concise-output instruction across training rounds, thereby internalizing the requirement for concise outputs into the model's own generation behavior. Specifically, as shown in Figure 1, we first obtain original generations from the unmodified input prompts , and produce the corresponding compressed generations by augmenting the same prompts with a concise-output instruction. If the compressed generations are on average shorter, we remove the instruction from the prompts and pair the resulting concise outputs with the original inputs as supervision signals to fine-tune the model. After training, we check whether this round has further reduced the average CoT length; if so, we repeat the above procedure until the model's compression capability approaches its limit. Requiring only simple prompt transformations, TokenDistill imposes minimal prerequisites and is thus readily applicable to diverse CoT scenarios.

Extensive experiments demonstrate that TokenDistill consistently outperforms state-of-the-art CoT compression approaches, with only minimal accuracy loss (an example is shown in Figure 2) . Moreover, we observe that, in some cases, TokenDistill can achieve substantial compression while even improving accuracy. We conjecture that this is partly because TokenDistill encourages the model to retain key reasoning steps while aggressively shortening the CoT. Free from the interference introduced by low-information details, the resulting focused reasoning may help the model arrive at the correct answer.

In summary, our contributions are threefold:

- We propose a novel training pipeline that enables the model to adaptively and iteratively distill CoT tokens without requiring any additional comparison samples or assistance from external models. The method has minimal prerequisites, requiring only simple prompt transformations, making it readily applicable to diverse settings in practical CoT scenarios.
- We introduce a continuous prompt transformation mechanism that internalizes the concise-output requirement into the model's behavior. By repeatedly adding and removing a concise-

output instruction over multiple rounds, the model gradually shifts from explicit prompting to intrinsic conciseness, achieving a near-optimal compression-performance trade-off.

- Extensive experiments show that our method significantly outperforms state-of-the-art CoT compression approaches. It achieves this with minimal accuracy loss and, in some cases, even yields slight accuracy improvements.

2 Related Work

CoT is a double-edged sword: while it improves the reasoning capabilities of LLMs, it also incurs substantial resource consumption. Therefore, researchers have sought to reduce the overhead of CoT.

Auxiliary-Model-Guided Compression. To achieve desirable compression capability, one intuitive approach is to leverage the power of external models. C3oT (Kang et al., 2025) enables the model to learn a short-CoT generation mode through conditioned training on long-short CoT pairs constructed by an external LLM. LLMLingua-2 (Pan et al., 2024) formulates prompt compression as a token classification task using token labels annotated by GPT-4 (Achiam et al., 2023). TokenSkip (Xia et al., 2025) constructs training data by pruning CoT tokens based on the importance ranking estimated by LLMLingua-2. A*-Thought (Xu et al., 2025) uses their proposed bidirectional importance scores (BIS) and A* search with a tailored cost function to assemble a short reasoning path based on high-quality data (Muennighoff et al., 2025) generated by DeepSeek-R1 (Guo et al., 2025). Although these studies can alleviate the overthinking problem to some extent, they rely on external model assistance. Consequently, their performance is closely tied to the quality of those external models. Moreover, such external intervention may introduce mismatches between the compression process and the target model's own reasoning patterns, thereby affecting reasoning quality, whereas our method depends entirely on the model's own capabilities and achieves strong compression performance.

Length-Aware Reinforcement Learning. In recent years, some researchers have begun to consider compression methods related to reinforcement learning (RL) (Schulman et al., 2017; Rafailov et al., 2023; Shao et al., 2024; Yu et al., 2025). Their main idea is to introduce a length-aware reward during RL training to penalize overly long responses. Kimi-k1.5 (RL) (Team et al., 2025) applies an explicit length penalty to encourage the model to favor shorter correct responses during training. O1-Pruner (Luo et al., 2025) designs a reward based on the reference model's length and accuracy for RL. L1 (Aggarwal & Welleck, 2025) introduces Length Controlled Policy Optimization (LCPO), which optimizes the model based on prompt-specified length and answer correctness. ConciseR (Song & Zheng, 2025) adopts a two-stage RL framework, where the model first improves its reasoning ability and then applies a length-aware GRPO objective to encourage concise reasoning. TokenSqueeze (Zhang et al., 2025) dynamically adjusts reasoning depth based on the correctness rate estimated from multiple sampled responses, and rewrites individual reasoning steps to effectively reduce the length of data for the training with its length-aware objective. Despite their effectiveness, RL-related methods generally incur higher training overhead than supervised learning, as they typically require additional reference signals or comparison samples to guide optimization. In addition, their performance is often sensitive to reward design and training stability, making them harder to optimize in practice. In contrast to these methods, our approach does not rely on reinforcement learning or carefully engineered loss functions. Instead, once the dataset is constructed, substantial performance gains can be achieved through simple supervised fine-tuning (SFT) alone.

3 Our Method

We propose a novel training pipeline that enables LLMs to distill their own CoTs into concise forms without any external model assistance. The pipeline proceeds over multiple rounds, each comprising four stages: data construction, pre-training evaluation, training, and post-training evaluation. Figure 3 illustrates the details of the k-th round. The pre-training evaluation stage determines whether the training stage should be carried out in the current round, while the post-training evaluation stage determines whether to proceed to the next round.

Round k takes three inputs: the model M_k obtained after the previous round of training, an initial dataset $\mathcal{D}_0$ used for training data construction, and an initial test set $\mathcal{T}_0$ used for post-training evaluation. The initial model M_0 is an off-the-shelf instruction-following pretrained LLM, such as

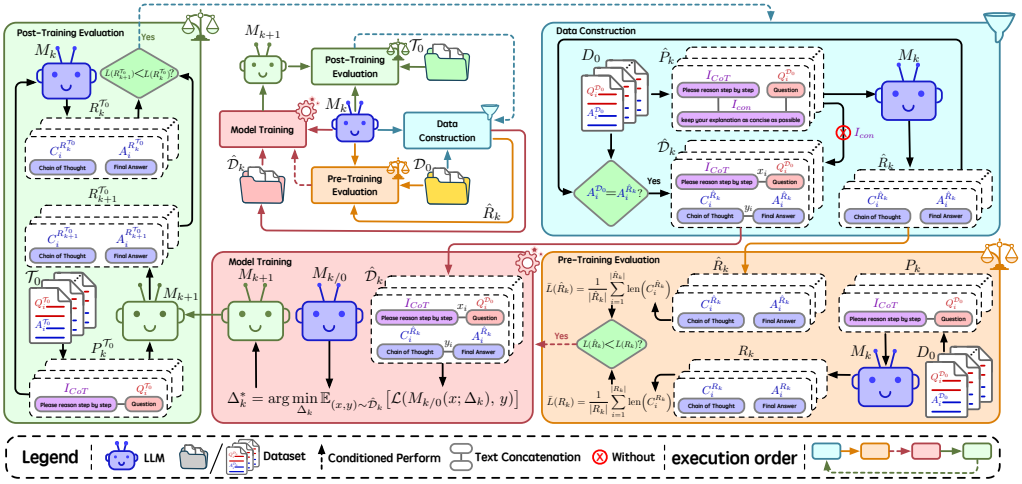

Figure 3: Implementation details of TokenDistill in round k. We construct the training dataset $\hat{\mathcal{D}}_k$ using generated concise responses $\hat{R}_k$ and original inputs $I_{CoT} \oplus Q^{\mathcal{D}_0}$. Model training is performed only when the concise responses $\hat{R}_k$ are shorter on average than the original responses R_k, i.e., $\bar{L}(\hat{R}_k) < \bar{L}(R_k)$. After training, we proceed to round $k+1$ only if the responses $R_{k+1}^{\mathcal{T}_0}$ generated by M_{k+1} on the evaluation set $\mathcal{T}_0$ are shorter on average than the responses $R_k^{\mathcal{T}_0}$ generated by M_k, i.e., $\bar{L}(R_{k+1}^{\mathcal{T}_0}) < \bar{L}(R_k^{\mathcal{T}_0})$. Here, $\oplus$ denotes text concatenation.

Qwen (Yang et al., 2024) or LLaMA (Grattafiori et al., 2024). The initial dataset $\mathcal{D}_0$ and test set $\mathcal{T}_0$ are derived from the training and test splits of the same source dataset (Cobbe et al., 2021; Hendrycks et al., 2021; Talmor et al., 2019), respectively.

3.1 Data Construction

In round k, we first extract all the questions $\{Q_i^{\mathcal{D}_0}\}_{i=1}^{|\mathcal{D}_0|}$ and their corresponding ground-truth answers $\{A_i^{\mathcal{D}_0}\}_{i=1}^{|\mathcal{D}_0|}$ from $\mathcal{D}_0$, where $|\mathcal{D}_0|$ denotes the number of samples in $\mathcal{D}_0$. For each question $Q_i^{\mathcal{D}_0}$, we concatenate it with the chain-of-thought instruction I_{CoT}, "Please reason step by step," and the concise-output instruction I_{con}, "keep your explanation as concise as possible," to form a single input prompt. Collecting all such prompts yields the input-prompt set $\hat{P}_k$ with the concise requirement:

$$\hat{P}_k = \left\{ I_{CoT} \oplus I_{con} \oplus Q_i^{\mathcal{D}_0} \right\}_{i=1}^{|\mathcal{D}_0|}, \tag{1}$$

where $\oplus$ denotes text concatenation and the hat notation $\hat{\cdot}$ denotes the concise variant. Then, we feed $\hat{P}_k$ into M_k to obtain the corresponding concise responses $\hat{R}_k$, each containing a CoT $C_i^{\hat{R}_k}$ and a final answer $A_i^{\hat{R}_k}$:

$$\hat{R}_k = \left\{ (C_i^{\hat{R}_k}, A_i^{\hat{R}_k}) \right\}_{i=1}^{|\mathcal{D}_0|} = M_k(\hat{P}_k). \tag{2}$$

Next, we filter out incorrect outputs by checking whether $A_i^{\mathcal{D}_0} = A_i^{\hat{R}_k}$ for each sample, thereby retaining only clean, successful CoT-compression samples to form the processed training dataset $\hat{\mathcal{D}}_k$:

$$\hat{\mathcal{D}}_k = \{(x_i, y_i)\}_{i=1}^{|\hat{\mathcal{D}}_k|} = \left\{ (I_{CoT} \oplus Q_i^{\mathcal{D}_0},\ C_i^{\hat{R}_k} \oplus A_i^{\hat{R}_k}) \;\middle|\; i \in \{1, \ldots, |\mathcal{D}_0|\},\ A_i^{\mathcal{D}_0} = A_i^{\hat{R}_k} \right\}. \tag{3}$$

where (x_i, y_i) denotes the i-th input-output training pair in $\hat{\mathcal{D}}_k$. Notably, $\hat{\mathcal{D}}_k$ excludes the concise-output instruction (i.e., I_{con}). We only keep I_{CoT} together with the question and the generated response. This design choice is a key component of our approach, as it enforces conciseness through data-driven self-distillation rather than explicit prompting. Appendix A.3 presents per-round sample retention statistics for one of our experimental settings.

3.2 Pre-Training Evaluation

Before starting training, we first verify whether the average token length of $\hat{R}_k$ is indeed reduced. To this end, we first construct the input-prompt set P_k without the concise requirement:

$$P_k = \left\{ I_{CoT} \oplus Q_i^{\mathcal{D}_0} \right\}_{i=1}^{|\mathcal{D}_0|}. \tag{4}$$

Then we feed P_k into M_k to obtain the original responses R_k:

$$R_k = \left\{ (C_i^{R_k}, A_i^{R_k}) \right\}_{i=1}^{|\mathcal{D}_0|} = M_k(P_k). \tag{5}$$

After obtaining R_k, we compare the average token lengths of the CoTs in $\hat{R}_k$ and R_k:

$$\bar{L}(\hat{R}_k) = \frac{1}{|\hat{R}_k|} \sum_{i=1}^{|\hat{R}_k|} \mathrm{len}\left(C_i^{\hat{R}_k}\right), \qquad \bar{L}(R_k) = \frac{1}{|R_k|} \sum_{i=1}^{|R_k|} \mathrm{len}\left(C_i^{R_k}\right), \tag{6}$$

where $\mathrm{len}(\cdot)$ denotes the token length. If $\bar{L}(\hat{R}_k) < \bar{L}(R_k)$, we proceed to the subsequent training stage using $\hat{\mathcal{D}}_k$; otherwise, no further training is performed and the iterative process terminates, as the model is considered to have reached its limit.

3.3 Model Training

Once the decision to proceed with training is made, we use the dataset $\hat{\mathcal{D}}_k = \{(x_i, y_i)\}_{i=1}^{|\hat{\mathcal{D}}_k|}$ from the Data Construction stage to perform supervised fine-tuning on the current-round model M_k, while employing LoRA (Hu et al., 2022) to reduce GPU memory consumption:

$$\Delta_k^* = \arg\min_{\Delta_k} \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}_k} \left[\mathcal{L}(M_k(x; \Delta_k),\, y) \right], \tag{7}$$

where $\mathcal{L}$ denotes the loss function that measures the discrepancy between the predicted output $M_k(x; \Delta_k)$ and the target output y, Δ_k denotes the trainable LoRA parameters, and Δ_k^* denotes the optimized parameters after training. After training, we obtain the model M_{k+1} for the next round:

$$M_{k+1} = M_k(\cdot; \Delta_k^*), \tag{8}$$

where $M_k(\cdot; \Delta_k^*)$ denotes the model obtained by equipping the current-round model M_k with the optimized LoRA parameters Δ_k^*.

In addition to the above training strategy that uses M_k as the base model, we also explore an alternative training strategy that uses M_0 as the base model:

$$M_{k+1} = M_0(\cdot; \Delta_k^*), \quad \Delta_k^* = \arg\min_{\Delta_k} \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}_k} \left[\mathcal{L}(M_0(x; \Delta_k),\, y) \right]. \tag{9}$$

We evaluate both training strategies and find that they achieve comparable performance. This suggests that the key to the method lies in constructing a dataset that the model can effectively understand and learn from. Meanwhile, the progressive data-construction process enables the model to distill tokens in a gradual, step-by-step manner.

3.4 Post-Training Evaluation

After training, we determine whether to proceed to the next round $k+1$ by evaluating whether the newly trained model M_{k+1} achieves improved performance. Specifically, we first extract all questions $\{Q_i^{\mathcal{T}_0}\}_{i=1}^{|\mathcal{T}_0|}$ and their corresponding answers $\{A_i^{\mathcal{T}_0}\}_{i=1}^{|\mathcal{T}_0|}$ from the test set $\mathcal{T}_0$. Subsequently, we concatenate $\{Q_i^{\mathcal{T}_0}\}_{i=1}^{|\mathcal{T}_0|}$ with I_{CoT}, "Please reason step by step," to construct the test input-prompt set $P_k^{\mathcal{T}_0}$:

$$P_k^{\mathcal{T}_0} = \left\{ I_{CoT} \oplus Q_i^{\mathcal{T}_0} \right\}_{i=1}^{|\mathcal{T}_0|}. \tag{10}$$

After that, we feed $P_k^{\mathcal{T}_0}$ into M_k and M_{k+1}, respectively, to obtain the corresponding responses:

$$R_k^{\mathcal{T}_0} = \left\{ (C_i^{R_k^{\mathcal{T}_0}}, A_i^{R_k^{\mathcal{T}_0}}) \right\}_{i=1}^{|\mathcal{T}_0|} = M_k(P_k^{\mathcal{T}_0}), \quad R_{k+1}^{\mathcal{T}_0} = \left\{ (C_i^{R_{k+1}^{\mathcal{T}_0}}, A_i^{R_{k+1}^{\mathcal{T}_0}}) \right\}_{i=1}^{|\mathcal{T}_0|} = M_{k+1}(P_k^{\mathcal{T}_0}). \tag{11}$$

Finally, we evaluate the CoT compression capability of M_{k+1} by checking whether

$$\bar{L}(R_{k+1}^{\mathcal{T}_0}) < \bar{L}(R_k^{\mathcal{T}_0}), \;\; \bar{L}(R_{k+1}^{\mathcal{T}_0}) = \frac{1}{|R_{k+1}^{\mathcal{T}_0}|} \sum_{i=1}^{|R_{k+1}^{\mathcal{T}_0}|} \text{len}(C_i^{R_{k+1}^{\mathcal{T}_0}}), \;\; \bar{L}(R_k^{\mathcal{T}_0}) = \frac{1}{|R_k^{\mathcal{T}_0}|} \sum_{i=1}^{|R_k^{\mathcal{T}_0}|} \text{len}(C_i^{R_k^{\mathcal{T}_0}}). \tag{12}$$

If the above condition holds, we proceed to the next round; otherwise, the iterative process terminates.

4 Experiments

4.1 Experimental Setup

Base Models. For the base models, we consider both native instruction-tuned models and a distilled reasoning model. Specifically, we use the Qwen2.5-Instruct (Yang et al., 2024) series and LLaMA3.1-8B-Instruct (Grattafiori et al., 2024) as native models, while DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025) serves as the distilled model.

Datasets. To evaluate the generalization of our method across answer formats and difficulty levels, we conduct experiments on three benchmarks: GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2024), and CommonsenseQA (CSQA) (Talmor et al., 2019). GSM8K and MATH-500 are open-ended mathematical reasoning datasets, while CSQA focuses on multiple-choice commonsense reasoning. Among the two mathematical benchmarks, GSM8K mainly contains elementary problems, whereas MATH-500 features more challenging ones.

Hardware, Framework, and Training Time. All experiments are implemented using PyTorch. Experiments with the Qwen2.5-Instruct series and LLaMA3.1-8B-Instruct are conducted on eight NVIDIA RTX 3090 GPUs, while those with DeepSeek-R1-Distill-Qwen-7B run on two NVIDIA RTX PRO 6000 GPUs. TokenDistill involves multiple training rounds and typically achieves strong performance after 3 – 5 rounds. Within each round, the average model training time is approximately 20 minutes for Qwen2.5-3B/7B-Instruct and LLaMA3.1-8B-Instruct, 1 hour for Qwen2.5-14B-Instruct, and 4 – 5 hours for DeepSeek-R1-Distill-Qwen-7B.

Inference Configurations. Our pipeline consists of four stages: data construction, pre-training evaluation, model training, and post-training evaluation. Only the model training stage involves parameter optimization, while the other three stages are inference-only. For these inference-only stages, we set the maximum generation length according to the model and benchmark. For the Qwen2.5-Instruct series and LLaMA3.1-8B-Instruct, `max_new_tokens` is set to 512 on GSM8K and 1024 on MATH, since MATH generally requires longer reasoning traces. For DeepSeek-R1-Distill-Qwen-7B, we set `max_new_tokens` to 2048 on GSM8K and 4096 on MATH to accommodate its longer reasoning outputs.

Training Settings. For the training stage, we apply LoRA (Hu et al., 2022) for supervised fine-tuning (Zheng et al., 2024) with `lora_target=all`, `lora_rank=8`, and `lora_alpha=16`. For optimization, we use the AdamW optimizer with a learning rate of 5×10^{-5}, together with a warmup ratio of 0.1 and cosine learning rate decay. The global batch size is set to 8 with 8 gradient accumulation steps, resulting in an effective batch size of 64. Training is conducted for 3 epochs in bfloat16 precision.

Metrics. We use three metrics to assess the compression performance of TokenDistill: Accuracy (Acc), Average Token Length (ATL), and Actual Ratio (AR). Accuracy measures the percentage of correctly answered samples. ATL denotes the average token length of the generated CoTs. AR is defined as the ratio between the ATL under the current method and that of the original baseline, where a smaller value indicates stronger compression. We report AR in decimal form.

4.2 Evaluation on Native Instruction-Tuned Models

To evaluate TokenDistill on native instruction-tuned models, we compare it with two state-of-the-art compression methods on MATH-500, GSM8K, and CSQA using the metrics introduced above. One is a prompt-based approach following Concise Chain-of-Thought (CCoT) (Renze & Guven, 2024), which appends a "be concise" instruction to the prompt. The other is TokenSkip (Xia et al., 2025),

Table 1: Comparison results for native instruction-tuned models on MATH-500, GSM8K, and CSQA.

Methods	Ratio	Qwen2.5-3B			Qwen2.5-7B			Qwen2.5-14B			LLaMA-3.1-8B		
		Acc	ATL	AR	Acc	ATL	AR	Acc	ATL	AR	Acc	ATL	AR
MATH-500													
Original	–	63.4	577.91	1.00	71.2	574.65	1.00	75.4	583.79	1.00	48.2	503.65	1.00
CCoT	–	66.2	524.33	0.91	70.2	533.50	0.93	75.0	538.37	0.92	50.8	459.41	0.91
TokenSkip	1.0	63.4	547.40	0.95	70.4	569.39	0.99	74.4	561.29	0.96	45.8	482.98	0.96
	0.9	60.4	501.77	0.87	71.4	496.17	0.86	74.2	511.47	0.88	45.4	435.33	0.86
	0.8	57.2	466.69	0.81	65.4	470.79	0.82	71.6	484.09	0.83	47.6	383.13	0.76
	0.7	52.6	442.57	0.77	63.4	421.34	0.73	69.0	453.58	0.78	44.4	349.30	0.69
	0.6	49.4	396.21	0.69	58.0	363.39	0.63	63.0	419.42	0.72	42.8	320.71	0.64
	0.5	43.8	343.50	0.59	53.8	317.86	0.55	61.0	357.69	0.61	38.0	286.75	0.57
TokenDistill	–	**59.6**	**381.56**	**0.66**	**70.8**	**375.32**	**0.65**	**72.8**	**322.51**	**0.55**	**45.2**	**312.80**	**0.62**
GSM8K													
Original	–	83.6	316.29	1.00	91.5	298.58	1.00	93.2	313.78	1.00	86.4	213.08	1.00
CCoT	–	83.0	252.98	0.80	91.5	251.89	0.84	93.8	277.53	0.88	84.7	155.33	0.73
TokenSkip	1.0	83.2	318.58	1.01	91.7	295.84	0.99	92.9	314.56	1.00	86.7	213.60	1.00
	0.9	83.2	262.30	0.83	91.0	254.76	0.85	93.4	269.65	0.86	86.1	198.01	0.93
	0.8	82.0	250.70	0.79	90.3	236.99	0.79	93.1	247.52	0.79	84.3	169.89	0.80
	0.7	80.4	233.03	0.74	89.5	216.56	0.73	93.4	218.73	0.70	82.5	150.12	0.70
	0.6	76.6	199.41	0.63	87.9	178.25	0.60	92.6	180.27	0.57	81.1	129.38	0.61
	0.5	74.6	169.48	0.54	85.8	151.82	0.51	91.1	157.21	0.50	78.2	113.05	0.53
TokenDistill	–	**82.0**	**145.68**	**0.46**	**90.6**	**169.41**	**0.57**	**93.6**	**159.81**	**0.51**	**84.2**	**149.18**	**0.70**
CSQA													
Original	–	73.3	253.10	1.00	79.9	265.40	1.00	81.5	223.00	1.00	71.3	194.20	1.00
CCoT	–	73.1	197.50	0.78	80.9	194.30	0.73	82.4	151.40	0.68	73.2	120.00	0.62
TokenSkip	1.0	73.7	254.26	1.00	80.9	270.90	1.02	87.7	211.69	0.95	74.9	192.60	0.99
	0.9	73.6	232.02	0.92	80.9	246.39	0.93	86.1	196.17	0.88	75.0	178.06	0.92
	0.8	73.8	210.76	0.83	82.0	222.77	0.84	85.6	173.26	0.78	75.0	161.45	0.83
	0.7	74.7	175.17	0.69	81.8	190.66	0.72	83.0	156.98	0.70	75.8	140.50	0.72
	0.6	73.1	153.25	0.61	80.6	164.42	0.62	78.7	137.34	0.62	76.0	122.99	0.63
	0.5	74.1	129.77	0.51	81.7	134.79	0.51	75.2	120.17	0.54	76.2	100.23	0.52
TokenDistill	–	**74.9**	**71.10**	**0.28**	**82.1**	**62.20**	**0.23**	**82.8**	**38.10**	**0.17**	**70.5**	**44.40**	**0.23**

which uses LLMLingua-2 (Pan et al., 2024) for token-importance estimation and supports explicit control over the compression level via an input ratio parameter. The quantitative results are presented in Table 1. For comparison, we select the TokenSkip setting with the smallest ATL among those whose accuracy is no more than 4 percentage points lower than the Original, and mark the selected settings in the table. As shown in Table 1, although CCoT achieves competitive accuracy, its compression effect remains limited. Compared with the selected TokenSkip settings, our method generally achieves lower AR while maintaining comparable or higher accuracy in most cases. Furthermore, we compute the average results across MATH-500, GSM8K, and CSQA. As shown in Figure 4, our method generally provides a more favorable accuracy-length trade-off than both CCoT and TokenSkip. In particular, across the Qwen models, it consistently achieves lower ATL than TokenSkip while also maintaining higher average accuracy. Compared with CCoT, our method yields substantially stronger compression while preserving competitive overall performance. On

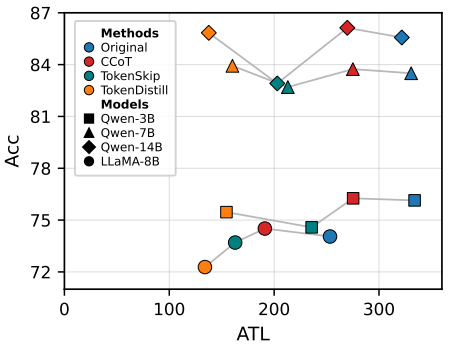

Figure 4: Acc–ATL trade-off across MATH-500, GSM8K, and CSQA.

Table 2: Comparison results for DeepSeek-R1-Distill-Qwen-7B on MATH-500, GSM8K

Method	MATH-500			GSM8K		
	Acc	ATL	AR	Acc	ATL	AR
Original	75.6	2519.03	1.00	86.5	1199.08	1.00
A*-Thought	73.6	1781.16	0.71	84.5	727.16	0.61
TokenSqueeze	81.6	1531.98	0.61	87.3	342.23	0.29
TokenDistill	**83.0**	**1221.44**	**0.48**	**90.6**	**313.74**	**0.26**

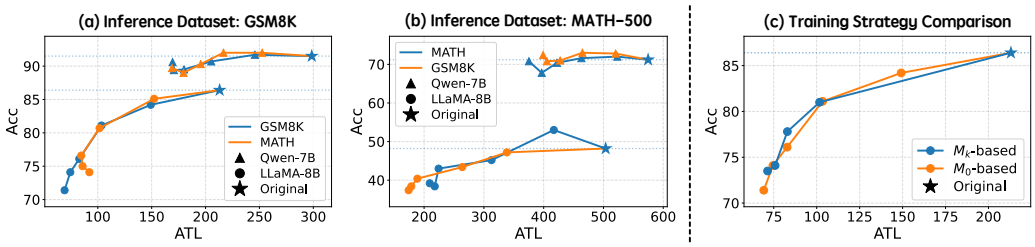

Figure 5: Cross-dataset generalization and training strategy comparison across training rounds.

LLaMA-3.1-8B-Instruct, our method still effectively reduces ATL relative to both CCoT and TokenSkip, without incurring a large loss in accuracy. Additionally, Appendices A.1, A.2, and A.4 present per-round results, an answer-only comparison, and a stability analysis across five random seeds, respectively.

4.3 Evaluation on Distilled Model

Distilled models tend to produce longer outputs, as their distillation data typically comes from strong reasoning models that generate longer CoTs. To further validate the effectiveness of TokenDistill on such models, we use DeepSeek-R1-Distill-Qwen-7B as the base model and compare TokenDistill with A*-Thought (Xu et al., 2025) and TokenSqueeze (Zhang et al., 2025) on MATH-500 and GSM8K. The quantitative results are reported in Table 2. As shown in the table, TokenDistill achieves the highest accuracy on both MATH-500 and GSM8K, while substantially reducing ATL compared with the original model and A*-Thought. Compared with TokenSqueeze, TokenDistill further reduces ATL from 1531.98 to 1221.44 on MATH-500 and from 342.23 to 313.74 on GSM8K, while also improving accuracy from 81.6 to 83.0 and from 87.3 to 90.6, respectively. These results demonstrate that TokenDistill remains effective even on distilled reasoning models with long CoT outputs. Additionally, an extended comparison with O1-Pruner is presented in Appendix A.5.

4.4 Cross-Dataset Generalization Across Training Rounds

To assess the cross-dataset generalization of our method, we train Qwen2.5-7B-Instruct and LLaMA-3.1-8B-Instruct on either MATH or GSM8K, and then evaluate each trained model on both MATH-500 and GSM8K. For clearer observation of the cross-round trends, we run five training rounds in this analysis without applying the stopping criterion. Figure 5(a) and (b) report Acc and ATL under two inference datasets: in Figure 5(a), all models are evaluated on GSM8K, while in Figure 5(b), all models are evaluated on MATH-500. In each subplot, the line color indicates the training dataset, the marker shape indicates the base model, the star denotes the original model before training, and each subsequent connected point corresponds to one additional training round. As shown in Figure 5, the overall compression trends remain largely consistent even when the training and inference datasets differ. In both settings, cross-dataset curves generally follow trajectories similar to the in-domain ones, suggesting that the gains from iterative compression are not restricted to the training dataset itself. This suggests that the model learns a transferable ability to identify and remove redundant tokens, rather than only fitting dataset-specific patterns. Overall, these results demonstrate the cross-dataset generalization capability of our method.

Table 3: Statistics of wrong-to-right cases after TokenDistill. Samples denotes the number of evaluation samples. W→R denotes cases where the original response is wrong but becomes correct after TokenDistill. Trunc and Non-Trunc divide W→R cases according to whether the original error is caused by premature truncation, and Ratio denotes the proportion of Non-Trunc cases. Here, 3B, 7B, and 14B denote Qwen2.5-Instruct models, while 8B denotes LLaMA3.1-8B-Instruct.

	GSM8K				MATH-500			
	3B	7B	14B	8B	3B	7B	14B	8B
Samples	1319	1319	1319	1319	500	500	500	500
W→R	42	45	38	68	28	32	33	42
Trunc	10	14	22	8	6	12	13	8
Non-Trunc	32	31	16	60	22	20	20	34
Ratio	0.76	0.69	0.42	0.88	0.79	0.63	0.61	0.81

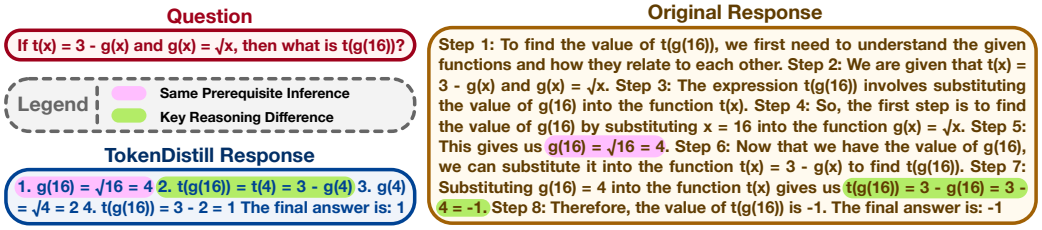

Figure 6: A qualitative example where TokenDistill corrects an originally wrong response from LLaMA3.1-8B-Instruct on MATH-500.

4.5 Comparison of Two Training Strategies

Using LLaMA3.1-8B-Instruct on GSM8K, we further compare the two training strategies in Section 3.3: the stage-wise strategy based on the previous-round model M_k, and the alternative strategy that always uses the initial model M_0 as the base model. As shown in Figure 5(c), both strategies reduce ATL over five training rounds. Their similar trajectories suggest that progressively generated concise data supports compression even without accumulating parameter updates across rounds.

4.6 Observations on Accuracy Gains After Compression

In addition to reducing response length, we observe that TokenDistill can sometimes improve answer accuracy. To analyze this phenomenon, we collect wrong-to-right (W→R) cases, where the original response is incorrect but becomes correct after TokenDistill. Since some W→R cases may simply result from avoiding premature truncation, Table 3 separates them into truncation (Trunc) and non-truncation (Non-Trunc) cases. The results show that, except for Qwen2.5-14B-Instruct on GSM8K, the Non-Trunc ratio exceeds 60%, suggesting that the gains are not solely caused by avoiding premature truncation. One such example for LLaMA3.1-8B-Instruct on MATH-500 is presented in Figure 6. The prerequisite value $g(16) = 4$ is first correctly computed in both responses. However, the original model then incorrectly treats $t(g(16))$ as $3 - g(16)$, thereby yielding -1, whereas the model after TokenDistill correctly evaluates it as $t(4) = 3 - g(4) = 1$. This example demonstrates that TokenDistill does not merely compress the original model output. Instead, it encourages the model to retain key reasoning steps while substantially shortening the CoT. Moreover, by reducing interference from low-information details, it may sometimes help the model reach the correct answer.

5 Conclusion

TokenDistill compresses CoTs through iterative self-distillation with continuous prompt transformation, internalizing concise generation without auxiliary models or additional comparison samples. Experiments on instruction-tuned and distilled reasoning models show substantial token reductions with competitive accuracy and occasional accuracy gains. Cross-dataset evaluations further suggest that the learned compression behavior transfers beyond the training benchmark.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation. We also used the models in TokenDistill to generate concise CoT responses to existing training questions. Responses with correct final answers were retained as training supervision, as described in Section 3. We have not used generative AI tools for any other tasks with required disclosure. Additionally, we used generative AI tools for sourcing and searching for information, identifying relevant literature, editing the paper to improve readability, adapting evaluation code for additional comparison experiments, and aggregating experimental results. We reviewed all AI-assisted work. The AI-assisted code and aggregated results were checked by the authors, and AI-suggested references were verified against the original sources. We take responsibility for the final content of this work, including any text, code, aggregated results, or other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We facilitate reproducibility by providing a detailed description of the TokenDistill pipeline in Section 3, including data construction, pre- and post-training evaluation, the stopping criteria, and the training objective. Section 4.1 specifies the datasets, base models, hardware, inference configurations, training hyperparameters, and evaluation metrics. Upon acceptance, we will publicly release the implementation, configuration files, and training, inference, and evaluation scripts used to produce the reported results.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Daniel Adiwardana, Minh-Thang Luong, David R. So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977*, 2020.

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Conference on Language Modeling*, 2025.

Tom 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pp. 1877–1901, 2020.

Cheng-Han Chiang and Hung-yi Lee. Over-reasoning and redundant calculation of large language models. In *Proceedings of the European Chapter of the Association for Computational Linguistics*, pp. 161–169, 2024.

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

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

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3oT: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24312–24320, 2025.

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

Ayeong Lee, Ethan Che, and Tianyi Peng. How well do LLMs compress their own Chain-of-Thought? a token complexity approach. In *ICML Workshop on Efficient Systems for Foundation Models*, 2025.

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

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

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, 2025.

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

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics*, pp. 963–981, 2024.

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

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

Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Eric Michael Smith, Y-Lan Boureau, and Jason Weston. Recipes for building an open-domain chatbot. In *Proceedings of the Association for Computational Linguistics*, pp. 300–325, 2021.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mingyang Song and Mao Zheng. Walk before you run! concise LLM reasoning via reinforcement learning. *arXiv preprint arXiv:2505.21178*, 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the Association for Computational Linguistics: Human Language Technologies*, pp. 4149–4158, 2019.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 6000–6010, 2017.

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

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

Xiaoang Xu, Shuo Wang, Xu Han, Zhenghao Liu, Huijia Wu, Peipei Li, Zhiyuan Liu, Maosong Sun, and Zhaofeng He. A*-Thought: Efficient reasoning via bidirectional compression for low-resource settings. In *Advances in Neural Information Processing Systems*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, pp. 11809–11822, 2023.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. DialoGPT: Large-scale generative pre-training for conversational response generation. In *Proceedings of the Association for Computational Linguistics: System Demonstrations*, pp. 270–278, 2020.

Yuxiang Zhang, Zhengxu Yu, Weihang Pan, Zhongming Jin, Qiang Fu, Deng Cai, Binbin Lin, and Jieping Ye. TokenSqueeze: Performance-preserving compression for reasoning LLMs. In *Advances in Neural Information Processing Systems*, 2025.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. In *International Conference on Learning Representations*, 2023.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the Association for Computational Linguistics: System Demonstrations*, 2024.

A Appendix

A.1 Single-round versus multi-round distillation

Table 4: Comparison of multi-round and single-round TokenDistill. Each cell reports Acc / ATL.

MATH-500						
Backbone	r_0 (Original)	r_1	r_2	r_3	r_4	r_5
Qwen2.5-3B	63.4 / 577.91	63.0 / 516.44	60.8 / 458.93	59.6 / 381.56	53.2 / 337.36	49.8 / 277.11
Qwen2.5-7B	71.2 / 574.65	72.0 / 523.10	71.6 / 463.10	70.4 / 422.21	67.8 / 396.74	70.8 / 375.32
Qwen2.5-14B	75.4 / 583.79	77.0 / 523.43	76.6 / 478.08	74.2 / 425.08	72.0 / 364.69	72.8 / 322.51
LLaMA-3.1-8B	48.2 / 503.65	53.0 / 416.94	45.2 / 312.80	43.0 / 224.25	38.4 / 217.81	39.2 / 209.50
GSM8K						
Backbone	r_0 (Original)	r_1	r_2	r_3	r_4	r_5
Qwen2.5-3B	83.6 / 316.29	83.6 / 238.43	82.0 / 145.68	75.7 / 108.51	58.3 / 101.72	51.7 / 101.90
Qwen2.5-7B	91.5 / 298.58	91.7 / 245.83	90.7 / 205.30	89.5 / 180.02	89.4 / 170.57	90.6 / 169.41
Qwen2.5-14B	93.2 / 313.78	94.3 / 266.80	93.6 / 197.70	93.6 / 159.81	92.9 / 133.23	91.3 / 127.69
LLaMA-3.1-8B	86.4 / 213.08	84.2 / 149.18	81.1 / 103.45	76.1 / 82.80	74.1 / 74.50	71.4 / 69.19

To analyze the contribution of iterative distillation to compression effectiveness, we report in Table 4 the complete per-round results on MATH-500 and GSM8K for four backbone models: Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct, and LLaMA-3.1-8B-Instruct. Each entry reports Acc / ATL. r_0 represents the original model without distillation, r_1 represents the model after a single round of distillation, and r_2–r_5 represent the models obtained through subsequent iterative distillation rounds. As shown in Table 4, later rounds provide substantial additional compression relative to r_1. On MATH-500, Qwen2.5-7B achieves an additional ATL reduction of 28.3% at r_5, with an accuracy decrease of only 1.2 percentage points. On GSM8K, Qwen2.5-3B, Qwen2.5-7B, and Qwen2.5-14B achieve additional ATL reductions of 38.9%, 31.1%, and 40.1% at r_2, r_5, and r_3, respectively, with corresponding accuracy decreases of 1.6, 1.1, and 0.7 percentage points. These results demonstrate the necessity of iterative distillation for achieving stronger compression.

A.2 Comparison with Answer-only baseline

To further investigate the effect of completely removing the CoT on accuracy, we evaluate an answer-only baseline on MATH-500 and GSM8K. Specifically, we instruct Qwen2.5-7B-Instruct and LLaMA-3.1-8B-Instruct to output only the final answers and compare their performance with standard CoT reasoning and TokenDistill. As shown in Table 5, each entry reports Acc / ATL. For the answer-only setting, ATL is denoted by "–" because no reasoning trace is generated. Separately measured complete answer-only outputs contain only 5.35–10.80 tokens on average across the four settings. The results show that although answer-only generation produces the shortest outputs, it causes a substantial drop in accuracy. On MATH-500, the accuracy of Qwen2.5-7B-Instruct and LLaMA-3.1-8B-Instruct decreases from 71.2% and 48.2% under standard CoT reasoning to 26.0% and 14.0%, respectively. On GSM8K, their accuracy similarly decreases from 91.5% and 86.4% to

23.0% and 15.8%, respectively. In contrast, TokenDistill reduces ATL by 30.0%–43.3% while largely preserving the accuracy of standard CoT reasoning. These results demonstrate that TokenDistill achieves a substantially better trade-off between accuracy and reasoning length than completely removing the CoT.

Table 5: Performance comparison among standard chain-of-thought (CoT) reasoning, answer-only generation, and TokenDistill. Each entry reports Acc / ATL.

Dataset	Method	Qwen2.5-7B	LLaMA-3.1-8B
MATH-500	Standard CoT	**71.2** / 574.65	**48.2** / 503.65
	Answer-Only	26.0 / –	14.0 / –
	TokenDistill	70.8 / **375.32**	45.2 / **312.80**
GSM8K	Standard CoT	**91.5** / 298.58	**86.4** / 213.08
	Answer-Only	23.0 / –	15.8 / –
	TokenDistill	90.6 / **169.41**	84.2 / **149.18**

A.3 Per-Round Data Filtering

Table 6: Per-round sample retention for TokenDistill using DeepSeek-R1-Distill-Qwen-7B on the MATH training set. In each round, one candidate chain of thought is generated for each of the 7,500 training examples.

Metric	r_1	r_2	r_3	r_4	r_5
Retained Samples	6,123	6,249	6,349	6,242	6,180
Retention Rate	81.6%	83.3%	84.7%	83.2%	82.4%

To quantify the amount of training data retained after correctness-based filtering, we examine sample retention over five rounds of TokenDistill using DeepSeek-R1-Distill-Qwen-7B on the MATH training set, where r_i denotes the i-th iteration. In each round, 7,500 candidate CoTs are independently generated and filtered according to whether their final answers are correct. Table 6 reports the number of samples retained after filtering and their proportion of the total candidate samples. As shown in Table 6, the number of retained samples ranges from 6,123 to 6,349 across rounds, corresponding to retention rates between 81.6% and 84.7%. Overall, sample retention remains relatively stable throughout the iterative process.

A.4 Stability Across Random Seeds

Table 7: Per-round results of TokenDistill across five different random seeds. Results are reported as mean $\pm$ standard deviation.

Round	Acc	ATL
r_1	91.96 ± 0.23	242.60 ± 0.91
r_2	90.40 ± 0.37	195.23 ± 3.12
r_3	90.17 ± 0.94	172.20 ± 2.86
r_4	89.48 ± 0.53	160.63 ± 2.55
r_5	89.34 ± 0.41	152.63 ± 4.08

To evaluate the stability of TokenDistill, we conduct five additional independent runs using Qwen2.5-7B-Instruct on GSM8K, each with a different random seed and proceeding through all five rounds. The results are presented in Table 7, where r_i denotes the i-th round of TokenDistill. As shown in Table 7, mean ATL decreases consistently over all five rounds across random seeds, reaching 152.63 ± 4.08 at r_5, a 48.88% reduction from the original ATL of 298.58, while retaining an accuracy of $89.34\% \pm 0.41\%$. These results demonstrate the stability of TokenDistill across random seeds.

A.5 Extended Evaluation on Reasoning Models

Table 8: Comparison of TokenDistill and O1-Pruner using Marco-o1-7B on GSM8K.

Method	Acc	ATL	AR
Original	88.70	541.23	1.00
O1-Pruner	**90.90**	348.40	0.64
TokenDistill	88.63	**243.74**	**0.45**

To further validate TokenDistill's compression effectiveness on reasoning models, we extend TokenDistill to Marco-o1-7B on GSM8K, with O1-Pruner as a comparison baseline. As shown in Table 8, TokenDistill reduces ATL from 541.23 to 243.74, corresponding to a 55.0% reduction relative to the original model, while accuracy decreases from 88.70% to 88.63%, a difference of 0.07 percentage points. Compared with O1-Pruner, TokenDistill achieves a 30.0% lower ATL, although its accuracy is 2.27 percentage points lower. These results further demonstrate the applicability of TokenDistill to reasoning models and its ability to achieve a favorable trade-off between CoT length and accuracy.

A.6 Limitation

Although TokenDistill effectively improves the model's ability to compress CoTs, its performance may be influenced by the number of available training samples, as it involves supervised fine-tuning. In TokenDistill, training samples are constructed by retaining outputs that are both concise and correct. Therefore, when only a limited number of samples pass the filter, either due to the limited size of the source dataset or the difficulty of the questions, constructing sufficiently large and diverse training data becomes more challenging, which may weaken subsequent fine-tuning. In future work, we plan to explore self-augmentation strategies for such settings, allowing the model itself to expand the training set from limited valid samples, thereby enabling more effective subsequent training.