AgShort: Improving Chain-of-Thought Compression by Agreement Maximization

Anonymous ACL submission

Abstract

Large language models (LLMs) trained to produce chain-of-thought (CoT) are known to be better at problem-solving. However, generating this extra reasoning text inevitably increases latency and inference cost, motivating research in efficient ways of compressing CoT. Conditioned Compressed CoT (C3oT) achieves this through data augmentation, training on both original and summarized versions of CoTs. However, works in other domains like computer vision have proposed ways of improving data augmentation. Agreement Maximization (AgMax) does so by adding the "agreement" between the model's representations of augmented inputs as a part of the loss. We elaborate on C3oT's methods by applying AgMax to representations of answer tokens between long and short CoTs, explicitly enforcing that answers should be invariant to reasoning length. We call this hybrid method AgShort, and found that our training can improve plain C3oT's performance by up to 9.0% on mathematics (AMC23), 3.8% on expert-level knowledge (GPQA Diamond), and 6.8% on tool-calling (BFCL V2 • Live). While improvements are stronger in some model families than others, we find the average effect of AgShort to be consistently positive across multiple tasks. Our code and data are available in anonymous github.

1 Introduction

Large language models (LLMs) have become increasingly prevalent, moving from simple virtual assistants to interfacing with real-world, time-sensitive systems. For this reason, it has also become increasingly important to train LLMs that can produce "chain-of-thought" (CoT) to improve their ability for problem-solving (Ho et al., 2022). These "thinking" LLMs are undoubtedly useful in complex and multi-step tasks. However, generating this extra reasoning text inevitably comes with increased latency and cost (Kumar et al., 2025).

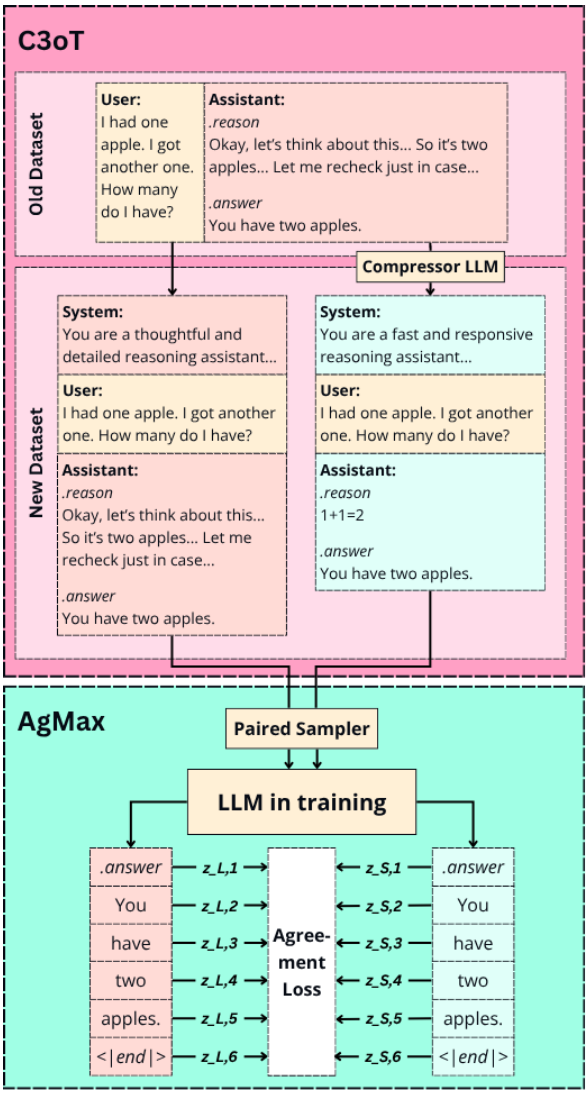

Figure 1: The AgShort pipeline, from C3oT's data augmentation to AgMax's agreement of answer tokens.

Due to this, compressing CoTs while preserving performance has been an active area of research.

Conditioned Compressed CoT (C3oT) (Kang et al., 2024) addresses this using data augmentation. They suggest training with both original (long) and summarized (short) versions of CoTs to implicitly teach the model the relationship between these versions, and preserve performance

even when prompted for shorter reasoning. Their results show this "simple and effective" method is better than training on summarized CoTs alone.

On the other hand, *Agreement Maximization* (AgMax) (Atienza, 2021) is a technique that aims to improve regular data augmentation by taking inspiration from representation learning. They suggest an additional term to the training loss based on a similarity measure (i.e. "agreement") of the model's representations, in logits, for two versions of the same input. Their results show this "simple constraint" improves plain data augmentation in computer vision and speech tasks.

This study elaborates on C3oT's data augmentation by applying AgMax to representations of answer tokens between long and short CoTs. We ask whether explicitly enforcing answers to be invariant to reasoning length can help train more efficient thinking LLMs. We call our training method AgShort, shown in Figure 1, and compare it with training on long CoTs, short CoTs, and plain C3oT. We investigate our method with AgMax set to different weights (0.1, 1.0, 10.0), and training across multiple model families (Phi-4-mini-instruct (Abdin et al., 2025), Qwen3-4B-Instruct-2507 (Yang et al., 2025), Olmo-3-7B-Instruct (Team Olmo: Allyson Ettinger et al., 2025)).

We observe that training with AgShort outperforms plain C3oT by up to 9.0% on mathematics with AMC23 (AMC), 3.8% on expert-level knowledge with GPQA Diamond (Rein et al., 2023), and 6.8% on tool-calling with BFCL V2 • Live (Patil et al., 2025) while preserving, and in some setups, even reducing CoT length. We find that while gains are fairly robust across weights, Phi-4 and Qwen3 benefit more strongly from our intervention than Olmo-3.

2 Methodology

2.1 Dataset Creation

We use the Dolci-Think-SFT-7B (Team Olmo: Allyson Ettinger et al., 2025) open-source CoT dataset as our base dataset. Inspired by Kopiczko et al. (2026), we filter this dataset down to only the first conversation turns, examples not exceeding 1,000 tokens and have non-empty CoT, and finally to a random subset of 1,000 examples.

As outlined in C3oT, a "compressor model" is used to summarize long CoTs from the base dataset. We employ Qwen3.6-27B (Yang et al., 2025) as a newer and open-weights alternative to their GPT-4. The model is prompted with example summaries (few-shot prompting) to help preserve the structure of the original CoTs while reducing verbosity (see Appendix A for the exact prompt). The final dataset, therefore, has four fields per example: the user instruction, original long CoT, summarized short CoT, and expected final answer.

2.2 Training

We compare 6 different training setups: long CoT, short CoT, C3oT, and C3oT with AgMax (AgShort) across 3 log-spaced weights (0.1, 1.0 and 10.0).

C3oT also outlines that long and short CoT examples differ by "conditioned" prompts, and inference is done with the "short condition". We elaborate their prompts into full system messages (e.g. "You are a thoughtful reasoning assistant...", "You are a fast reasoning assistant..."), and train long CoT and short CoT setups with the short condition as well to preserve fairness during inference. Additionally, models are trained to respond with the format "`.reason <CoT> .answer <final answer>`" which is crucial for identifying the answer tokens for AgMax (see Appendix B for the full chat templates).

We perform supervised fine-tuning (SFT) using QLoRA with Unsloth, following their recommended starting hyperparameters (Unsloth). Referencing findings by Kopiczko et al. (2026) that high epoch training with a small dataset is optimal for CoT SFT, we set a fixed training budget of 500 gradient updates with a batch size of 16, corresponding to approximately 8 epochs of training (see Appendix C for all hyperparameters).

We investigate training on Phi-4-mini-instruct by Microsoft (Abdin et al., 2025), Qwen3-4B-Instruct-2507 by Alibaba (Yang et al., 2025), and Olmo-3-7B-Instruct by Ai2 (Team Olmo: Allyson Ettinger et al., 2025), with our considerations for choosing these models being: (1) instruction-tuned with no prior CoT SFT, (2) in the 3-to-8B-parameter range due to hardware constraints, (3) relatively recent; released in 2025 and onwards, (4) tokenizes `.answer` as a single token (for AgMax), which avoids training for additional tokens with SFT.

2.3 Agreement Maximization

Unlike in computer vision, there is no one single "label" for next-token prediction. We apply AgMax to all final answer tokens, analogous to the shared label between two augmented images in their original study (Atienza, 2021).

Paired-random sampling. To perform agreement, augmented samples must first be batched together. We implement a custom sampler which takes a unique instruction-answer pair from C3oT's mixed dataset, and samples both the long and short CoT versions together.

Search for answer tokens. In each example, we perform a vectorized search for the `.answer` token, and the last end-of-sequence token (e.g. `<|end|>` for Phi-4). Note that these indices make up a range that captures all final answer tokens in any given example following our chat template.

Agreement by cosine similarity loss. Since long and short CoT versions have identical final answers, we expect all answer tokens in one to also exist in the other (albeit, at an offset due to differences in prompt and CoT). For each pair of corresponding answer tokens, we compute cosine similarity loss on their logits z_L and z_S. Then, take the average across all pairs A as the final agreement loss of the two corresponding examples:

$$L_{ag} = \frac{1}{A}\sum_{i=1}^{A}(1 - \cos(z_{L,i}, z_{S,i})) \tag{1}$$

The complete loss for each pair of long and short CoT examples is therefore:

$$L = L_{pred} + \lambda L_{ag} \tag{2}$$

where L_{pred} is the standard next-token prediction loss of both examples, and L_{ag} is the agreement loss between examples weighted by a constant λ.

2.4 Evaluation

We evaluate our models on modern benchmarks across 3 distinct domains. For mathematics, we use AMC23, composed of problems from the American Mathematics Competition (AMC). For knowledge, we use GPQA Diamond, composed of natural science questions authored by PhD-level experts (Rein et al., 2023). Finally, for tool-calling, we use BFCL V2 • Live, composed of problems requiring external function calls that reflect real-world scenarios (Patil et al., 2025). Due to the smaller size of AMC23 and GPQA Diamond, we take the average across 5 trials (seeded from 0 to 4) for these benchmarks.

To allow for automatic checking, final answers must follow a specified format. We enforce this on our models through prompting (e.g. additional instructions, few-shot examples, etc.) (see Appendix D for the full evaluation prompts).

We report 3 metrics. Accuracy is simply exact-match correctness for AMC23 and GPQA Diamond, and the score on the official evaluation API for BFCL V2 • Live. Output length is the average number of generated tokens, including the CoT and the final answer. Efficiency is a combined metric for overall reasoning efficiency. We use OckScore by Du et al. (2025) which enforces an ordinal ranking where models that are more correct but think longer (e.g. 50% for 100 tokens) should score higher than models that are more wrong but think proportionally shorter (e.g. 5% for 10 tokens). This score is a log-penalty on accuracy described by:

$$S_{ock} = A - \alpha \log\left(\frac{T}{\beta}\right) \tag{3}$$

where A is accuracy from 0 to 100, T is the model's average output length in tokens, α is the penalty weight set to 10 as recommended, while β is a normalization constant set to 100 to keep the penalty term negative.

3 Main Results

3.1 Short CoT and C3oT

From Table 1, we first observe that short CoT on its own outperforms long CoT on our math and knowledge benchmarks. It achieves an average of ~3% higher accuracy while using less than half as many tokens. This suggests that, with well-formed summaries, training with only short CoTs does not necessarily degrade performance, and can even improve it.

Next, we observe that while C3oT improves on short CoT's accuracy across all benchmarks, the effect is extremely marginal (<0.6%) and comes with an increased output length that leads to overall reasoning efficiency being worse in AMC23 and GPQA Diamond.

3.2 Comparing Benchmarks

On average, we observe that models trained with AgShort consistently outperform models trained with plain C3oT on all 3 benchmarks, and are best performing in both mathematics and tool-calling.

In AMC23, AgShort models show ~5% higher accuracy and efficiency compared to C3oT, and are the best performing models by that same margin. In GPQA Diamond, improvements are more marginal at ~1% higher accuracy. Increases in output length place overall gains in efficiency at

Model	Setup	AMC23 (5 trials)			GPQA Diamond (5 trials)			BFCL V2 • Live		
		Acc. (%) ↑	Len. (tok.) ↓	Eff. (S_{ock}) ↑	Acc. (%) ↑	Len. (tok.) ↓	Eff. (S_{ock}) ↑	Acc. (%) ↑	Len. (tok.) ↓	Eff. (S_{ock}) ↑
Phi-4-mini-instruct	Long CoT	30.0	1,386	18.6	**29.0**	656	20.8	43.8	460	37.1
	Short CoT	25.0	589	17.3	25.2	**262**	21.0	**47.7**	**159**	**45.7**
	C3oT	22.5	639	14.5	22.8	266	18.6	41.8	202	38.7
	+AgMax-0.1	(+8.0) 30.5	(-42) 597	(+8.3) 22.7	(+1.0) 23.8	(+10) 276	(+0.8) 19.4	(+1.3) 43.1	(+5) 207	(+1.2) 39.9
	+AgMax-1.0	**(+8.5) 31.0**	**(-97) 542**	**(+9.2) 23.7**	(+3.8) 26.7	(+1) 267	**(+3.8) 22.4**	(+0.9) 42.7	(-2) 200	(+0.9) 39.6
	+AgMax-10.0	(+3.0) 25.5	(-90) 549	(+3.7) 18.1	(+2.6) 25.5	(+11) 277	(+2.4) 21.0	(+4.0) 45.7	(+7) 209	(+3.8) 42.5
Qwen3-4B Instruct-2507	Long CoT	47.5	2,078	34.3	28.0	1781	15.5	**61.2**	707	52.7
	Short CoT	45.5	**1,302**	34.4	34.9	**770**	**26.0**	46.7	182	44.1
	C3oT	48.5	1,603	36.5	34.0	930	24.4	53.5	181	51.0
	+AgMax-0.1	(+5.0) 53.5	(+10) 1,613	(+5.0) 41.4	(+0.8) 34.9	(+57.1) 987	(+0.5) 24.9	(+2.7) 56.2	(-3) 177	(+2.7) 53.7
	+AgMax-1.0	(+8.0) 56.5	(-8) 1,595	(+8.0) 44.5	**(+1.3) 35.3**	(+64.3) 995	(+0.9) 25.3	(+4.6) 58.2	(-10) 171	(+4.9) 55.8
	+AgMax-10.0	**(+9.0) 57.5**	(-101) 1,502	**(+9.3) 45.7**	**(+1.3) 35.3**	(-4.1) 926	(+1.2) 25.6	(+6.8) 60.4	**(-14) 167**	**(+7.2) 58.2**
Olmo-3-7B-Instruct	Long CoT	23.5	2,496	9.5	18.3	1982	5.3	**47.0**	810	37.9
	Short CoT	41.5	**916**	**31.9**	24.8	**378**	19.0	38.9	157	37.0
	C3oT	42.5	1,496	30.8	28.6	505	21.6	39.2	**145**	37.6
	+AgMax-0.1	(0.0) 42.5	(+50) 1,546	(-0.2) 30.6	**(+1.2) 29.8**	(+36) 541	**(+0.9) 22.5**	(+0.2) 39.4	(+7) 152	(0.0) 37.6
	+AgMax-1.0	**(+1.0) 43.5**	(+184) 1,680	(+0.5) 31.3	(-0.9) 27.7	(+69) 575	(-1.5) 20.1	(+4.2) 43.4	(+12) 157	**(+3.8) 41.4**
	+AgMax-10.0	**(+1.0) 43.5**	(+72) 1,569	(+0.8) 31.6	(-1.2) 27.4	(+128) 634	(-2.2) 19.4	(-1.2) 38.0	(+2) 148	(-1.2) 36.3
Average	Long CoT	33.7	1,987	20.7	25.1	1473	13.4	**50.6**	659	42.5
	Short CoT	37.3	**936**	27.6	28.3	**470**	21.5	44.5	**166**	42.3
	C3oT	37.8	1,246	26.9	28.5	567	21.0	44.8	176	42.4
	+AgMax-0.1	(+4.3) 42.2	(+6) 1,252	(+4.3) 31.2	(+1.0) 29.5	(+34) 602	(+0.8) 21.7	(+1.4) 46.2	(+3) 179	(+1.3) 43.7
	+AgMax-1.0	**(+5.8) 43.7**	(+26) 1,272	**(+5.7) 32.6**	**(+1.4) 29.9**	(+45) 612	**(+1.0) 22.0**	(+3.3) 48.1	(0) 176	**(+3.2) 45.6**
	+AgMax-10.0	(+4.3) 42.2	(-40) 1,207	(+4.5) 31.4	(+0.9) 29.4	(+45) 612	(+0.5) 21.5	(+3.2) 48.0	(-1) 174	**(+3.2) 45.6**

Table 1: Summary of results across models and benchmarks. In parentheses are the absolute differences of AgShort setups compared to C3oT. In bold are the best scoring setups per model per metric.

$\sim$0.8%. In BFCL V2 • Live, improvements in both accuracy and efficiency are at $\sim$3%, but fall behind long CoT in accuracy by $\sim$2%.

These results show that the effects of AgShort, while a consistent improvement, are not symmetric across tasks. Effects seem to be greater for tasks requiring more multi-step reasoning, whether explicit (mathematics) or implicit (tool-calling). But are less for tasks requiring extensive background knowledge (expert-level natural science).

3.3 Comparing Models

Our results also show some variance in the effects of AgShort for different models. With Phi-4-mini-instruct, improvements in efficiency are consistent and substantially above average for AMC23 (up to 9.22%) and GPQA Diamond (up to 3.82%). With Qwen3-4B-Instruct-2507, improvements in efficiency are also consistent and are the highest overall for AMC23 (up to 9.28%) and BFCL V2 • Live (up to 7.19%). With Olmo-3-7B-Instruct, effects are marginal and even negative in some setups. The worst setups being in GPQA Diamond with -2.19%, and in BFCL V2 • Live with -1.23%.

This suggests that architectural or pre-training differences make AgShort weaker on Olmo-3. However, similar high gains in both Phi-4 and Qwen3 suggest that our intervention does have some consistent effect across models.

3.4 Comparing Weights

On average, we observe that AgShort consistently improves accuracy and efficiency across weight values for AgMax that are orders of magnitude apart (e.g. 0.1 and 10.0). The general trend seems to be that a weight of 1.0 offers the best improvements. However, there are multiple exceptions. For example, Qwen3 seem to consistently improve with higher AgMax weights across all benchmarks. Meanwhile, improvements in Phi-4 for BFCL V2 • Live are at their worst at a weight of 1.0.

These results suggest that our implementation of AgMax is largely stable across weights and generally offers best performance when set to 1.0. But with the higher gains seen at Qwen3 on a weight of 10.0, tuning this hyperparameter proves to be highly valuable.

4 Conclusions

We find that applying AgMax on C3oT's data augmentation can substantially improve CoT compression. Our hybrid method, AgShort, consistently outperforms training with plain C3oT on tasks requiring multi-step reasoning like mathematics and tool-calling. These gains are fairly robust across weights and model families, though vary by model, with Phi-4 and Qwen3 benefiting more strongly than Olmo-3. Overall, we demonstrate that AgShort is a simple but effective extension of C3oT for training more efficient thinking LLMs.

Limitations

Our study is limited to 3 models in the 3-to-8B range due to hardware constraints. We were also limited to smaller benchmarks due to the extensive wall-clock time required to inference with thinking LLMs. We welcome studies investigating AgShort for training larger models from other families, as well as larger benchmarks from other domains.

References

math-ai/amc23 · datasets at hugging face. https://huggingface.co/datasets/math-ai/amc23. Accessed: 2026-5-17.

Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, Piero Kauffmann, Yash Lara, Caio César Teodoro Mendes, Arindam Mitra, Besmira Nushi, Dimitris Papailiopoulos, Olli Saarikivi, Shital Shah, Vaishnavi Shrivastava, and 4 others. 2025. Phi-4-reasoning technical report.

Rowel Atienza. 2021. Improving model generalization by agreement of learned representations from data augmentation.

Randall Balestriero and Yann LeCun. 2022. Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations.

Zheng Du, Hao Kang, Song Han, Tushar Krishna, and Ligeng Zhu. 2025. OckBench: Measuring the efficiency of LLM reasoning.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2022. Large language models are reasoning teachers.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2024. C3oT: Generating shorter Chain-of-Thought without compromising effectiveness.

Dawid J Kopiczko, Sagar Vaze, Tijmen Blankevoort, and Yuki M Asano. 2026. Data repetition beats data scaling in long-CoT supervised fine-tuning.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. 2025. OverThink: Slowdown attacks on reasoning LLMs.

Zichao Li, Jie Lou, Fangchen Dong, Zhiyuan Fan, Mengjie Ren, Hongyu Lin, Xianpei Han, Debing Zhang, Le Sun, Yaojie Lu, and Xing Yu. 2026. Tackling length inflation without trade-offs: Group relative reward rescaling for reinforcement learning.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

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.

Team Olmo: Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, and 48 others. 2025. Olmo 3.

Raghuveer Thirukovalluru, Xiaolan Wang, Jun Chen, Shuyang Li, Jie Lei, Rong Jin, and Bhuwan Dhingra. 2024. SumCSE: Summary as a transformation for contrastive learning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3577–3588, Stroudsburg, PA, USA. Association for Computational Linguistics.

Unsloth. Lora fine-tuning hyperparameters guide.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding.

Yibo Wang, Haotian Luo, Huanjin Yao, Tiansheng Huang, Haiying He, Rui Liu, Naiqiang Tan, Jiaxing Huang, Xiaochun Cao, Dacheng Tao, and Li Shen. 2025. R1-Compress: Long chain-of-thought compression via chunk compression and search.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report.

A Compression Prompt

The following is the prompt we used on Qwen3.6-27B to summarize long CoTs into short CoTs. We include two CoT examples from Dolci-Think-7B and corresponding hand-written summaries to help preserve the structure of the original CoTs.

```
Simplify the following THOUGHT PROCESS, only
↪  including the key information in each step.
Do NOT add additional information that is not
↪  included in the original THOUGHT PROCESS.
Match the style and flow of the original THOUGHT
↪  PROCESS.
Respond with ONLY the SIMPLIFIED THOUGHT PROCESS
↪  and no other text.

Example THOUGHT PROCESS:
Okay, I need to write a Python program that
↪  calculates the area of a circle given its
↪  radius. Let me think about the steps here.

First, the program has to read input from
↪  standard input. The input is a floating-point
↪  number, which is the radius. So I'll use
↪  input() function to get that. But since it's
↪  a float, I have to convert it. Like, radius =
↪  float(input()), right?

Then, the formula for the area is π times radius
↪  squared. Oh right, I need to import the math
↪  module to get π. So I should import math and
↪  use math.pi. Alternatively, maybe from math
↪  import pi? But to be safe, maybe just use
↪  math.pi.

So the area would be math.pi * (radius ** 2).
↪  Then I need to output that. The output should
↪  be a floating-point number. So print the area
↪  with print.

Wait, but how does the output look exactly? The
↪  problem says to output a single
↪  floating-point number. So just print(area)
↪  should work, since the calculation would
↪  already be a float.

Let me check possible issues. What if the input
↪  is something like 0? Then the area is zero,
↪  which is okay. And negative radius? Hmm, the
↪  problem probably assumes a valid input, so
↪  maybe we don't have to handle that. The
↪  problem statement says the input is the
↪  radius, so maybe it's okay to proceed.

Putting it all together:

First line: import math.

Then read the radius: r = float(input())

Calculate area: area = math.pi * r**2

Then print(area).

Wait, but when I print, maybe I should format it
↪  to ensure it's a float? Like, if the user
↪  enters an integer as input, but the output
↪  must be a float. Well, the calculation will
↪  result in a float anyway, so printing it
↪  directly should be okay. Let me see. For
↪  example, if radius is 2, area would be
↪  math.pi * 4, which is like 12.566..., so
↪  printed as a float.

So the code structure would be:

import math

r = float(input())
print(math.pi * r ** 2)

Wait, that's even more concise. Because we can do
↪  it in two lines. No need to store the area in
↪  a variable, unless necessary. The problem
↪  doesn't require that, so this should work.

Hmm, but sometimes when you have to output a
↪  float, maybe there's a case where the result
↪  is an integer? Like radius 0, then it's 0.0?
↪  But in Python, 0.0 is still a float. So the
↪  output would be correct as is.

Testing with the input 5, the area is π*25 ≈
↪  78.5398163397..., so the program would print
↪  that. So this should be correct.

I think that's all. Let me write the code as
↪  such.

Example SIMPLIFIED THOUGHT PROCESS:
I need to write a Python program that calculates
↪  the area of a circle given its radius.

First, I need to read the radius from standard
↪  input. Since it's a float, I need to convert
↪  it. So radius = float(input()).

The formula for the area is π times radius
↪  squared. I need to import math first then use
↪  math.pi. So area = math.pi * (radius ** 2).

Then, since the output should be a float and
↪  calculation results in a float, I just need
↪  to print that value with print(area).

Let me check for possible issues. Input is 0.0?
↪  then area is 0.0, which is okay. Input is
↪  negative? Problem probably assumes valid
↪  input. Input is integer? math.pi is a float
↪  so output will still be a float.

Example THOUGHT PROCESS:
The problem states that an equilateral triangle
↪  has a side length of 20. A square has the
↪  same perimeter as this triangle, and I need
↪  to find the area of the square. The options
↪  are 25, 400, 225, 60, and 100.

First, since it's an equilateral triangle, all
↪  sides are equal. So, each side is 20. The
↪  perimeter of the triangle is three times the
↪  side length. So, perimeter P_triangle = 3 *
↪  20 = 60.

Now, the square has the same perimeter, so
↪  perimeter of square P_square = 60.

A square has four equal sides, so each side of
↪  the square is the perimeter divided by 4. So,
↪  side length s = 60 / 4 = 15.

The area of a square is side squared, so area A =
↪  s^2 = 15^2 = 225.

Looking at the options, 225 is one of them, which
↪  is choice (C).

I should double-check. Triangle perimeter: 3
↪  sides * 20 = 60. Square perimeter: 60. Each
↪  side of square: 60 / 4 = 15. Area: 15 * 15 =
↪  225. Yeah, that seems correct.
```

```
Option (B) is 400, which is 20 squared, but
↪  that's not relevant here. Option (A) 25 is 5
↪  squared, (D) 60 is the perimeter, not area,
↪  (E) 100 is 10 squared. So, 225 makes sense.

The triangle is equilateral with side 20,
↪  perimeter 60. Square same perimeter, side 15,
↪  area 225. I think that's it.

Example SIMPLIFIED THOUGHT PROCESS:
I need to find the area of a square whose
↪  perimeter is equal to an equilateral
↪  triangle.

The triangle has a side length of 20, and since
↪  it's equilateral, the perimeter is 3 * 20 =
↪  60.

So the square has the same perimeter of 60.

A square has four equal sides. So, the square's
↪  side length is 60 / 4 = 15.

The area of a square is side squared, so 15^2 =
↪  225.

In the given options, 225 is choice (C).

Your THOUGHT PROCESS:
{}

Your SIMPLIFIED THOUGHT PROCESS:
```

B Chat Templates

The following are the chat templates we used in training Phi-4-mini-instruct. Templates for other models differ slightly, based on their original chat template and control tokens.

B.1 Long CoT template

```
<|system|>\
You are a thoughtful and detailed reasoning
↪  assistant.
Your job is to answer questions and instructions
↪  with a thorough and comprehensive thought
↪  process.<|end|>\
<|user|>\
{user}<|end|>\
<|assistant|>\
.reason
{reasoning}

.answer
{answer}<|end|>\
```

B.2 Short CoT template

```
<|system|>\
You are a fast and responsive reasoning
↪  assistant.
Your job is to answer questions and instructions
↪  with a concise and to-the-point thought
↪  process.<|end|>\
<|user|>\
{user}<|end|>\
<|assistant|>\
.reason
{reasoning}

.answer
{answer}<|end|>\
```

C Hyperparameters

C.1 Training

We train our models with QLoRA using Unsloth, following their recommended starting settings for rank (16), alpha (32), learning rate (2e-4), warmup ratio (0.1), scheduler type (cosine), and effective batch size (16). Meanwhile, we reference C3oT for their optimizer type (8-bit AdamW), optimizer constants (β_1=0.9, β_2=0.999, ϵ=1e-6), and weight decay (0.001) (Kang et al., 2024).

C.2 Sampling

We follow Qwen3's recommended sampling parameters, using a temperature of 0.7, top-p of 0.8, top-k of 20, min-p of 0, and presence penalty of 1.5. Additionally, we constrain model outputs to a max sequence length of 3,000 tokens. Since training is on samples with lengths that are only up to around 1,000 tokens, we consider this a generous margin for generation, and restricts overly-long looping CoTs.

D Evaluation Prompts

The following are the prompts we used to constrain the final answer format of our models for automated checking:

D.1 AMC23

```
<|system|>\
You are a fast and responsive reasoning
↪  assistant.
Your job is to answer questions and instructions
↪  with a concise and to-the-point thought
↪  process.<|end|>\
<|user|>\
Solve the math problem below. Respond only with
↪  the final numeric answer inside \\boxed{{}}.
↪  No solutions or explanations. No units or
↪  currencies.

Example Question 1: Natalia sold clips to 48 of
↪  her friends in April, and then she sold half
↪  as many clips in May. How many clips did
↪  Natalia sell altogether in April and May?
Example Answer 1: \\boxed{{72}}

Example Question 2: Weng earns $12 an hour for
↪  babysitting. Yesterday, she just did 50
↪  minutes of babysitting. How much did she
↪  earn?
Example Answer 2: \\boxed{{10}}

Your Question: {user}
Your Answer: <|end|>\
<|assistant|>\
```

D.2 GPQA Diamond

```
<|system|>\
You are a fast and responsive reasoning
↪  assistant.
```

```
Your job is to answer questions and instructions
↪  with a concise and to-the-point thought
↪  process.<|end|>\
<|user|>\
Choose the best option that answers the question
↪  below. Respond only with the capital letter
↪  of your final answer inside \\boxed{{}}. No
↪  solutions or explanations. No other symbols.

Example Question 1: Which of the following is the
↪  body cavity that contains the pituitary
↪  gland?
A. Ventral
B. Dorsal
C. Buccal
D. Cranial
Example Answer 1: \\boxed{{D}}

Example Question 2: Colors in a soap bubble
↪  result from light
A. interference
B. deflection
C. refraction
D. reflection
Example Answer 2: \\boxed{{A}}

Your Question: {user}
Your Answer: <|end|>\
<|assistant|>\
```

D.3 BFCL V2 • Live

For BFCL V2 • Live, we did not include examples, but simply formatting instructions appended at the end of the available tools. The following is the prompt for an example user query:

```
<|system|>\
You are a fast and responsive reasoning
↪  assistant.
Your job is to answer questions and instructions
↪  with a concise and to-the-point thought
↪  process.<|end|>\
<|user|>\
What is the result of dividing 100 by 25?
Here is a list of functions that you can invoke:
- historical_exchange_rates: Fetch historical
↪  exchange rate data for a specific date using
↪  the RapidAPI service. Parameters:
  - date (string): The date for which to retrieve
  ↪  exchange rate data, formatted as
  ↪  'YYYY-MM-DD'.
- hull_moving_average: Calculates the Hull
↪  Moving Average (HMA) indicator for a given
↪  cryptocurrency market symbol using the
↪  Qvantana API. Parameters:
  - exchange (string): Name of the cryptocurrency
  ↪  exchange (e.g., 'binance'). Default is
  ↪  'binance'.
  - market (string): Type of the market (e.g.,
  ↪  'spot', 'usdt-perpetual'). Default is
  ↪  'spot'.
  - symbol (string): Trading symbol of the
  ↪  cryptocurrency (e.g., 'BTCUSD'). Default
  ↪  is 'btcusdt'.
  - interval (string): Time interval for the HMA
  ↪  calculation (e.g., '1d', '1h'). Default is
  ↪  '1m'.
  - is_from (string, optional): Start time for
  ↪  fetching data in UNIX timestamp format,
  ↪  default is '1683895800'.
  - backtracks (integer, optional): Number of
  ↪  periods to look back, default is 1.
  - length (integer, optional): Length of the
  ↪  HMA, default is 9.
Respond only with a list of function calls in
↪  Python format. No explanations or any other
↪  text. Example: [func1(x=1, y=2, ...),
↪  func2(), ...]
If there are no appropriate functions to help the
↪  task, simply respond with an empty list.
↪  Example: []<|end|>\
<|assistant|>\
```

E Related Work

The common post-training pipeline for thinking LLMs is supervised fine-tuning (SFT) on CoT examples followed by reinforcement learning (RL) (Yang et al., 2025)(Team Olmo: Allyson Ettinger et al., 2025). Some amount of "length inflation" is expected in the RL stage (Li et al., 2026), making interventions in the SFT stage fundamental and the focus of a large number of studies. TokenSkip by Xia et al. (2025) analyzes and prunes semantically less important tokens from CoT examples. R1-Compress by Wang et al. (2025) performs chunk-wise compression on CoT examples followed by search and selection.

Similarly, C3oT by Kang et al. (2024) works on the SFT stage by training on both long and short CoT examples (data augmentation). However, they train on task-specific datasets so final models can only reliably reason on a specific task (i.e. math or commonsense). Additionally, with the fast-moving field of LLM research, the models (Llama 2) and benchmarks (GSM8K, ECQA) they studied are largely considered outdated and saturated. Finally, while their goal is to improve short CoT performance by relating examples with long CoT in training, they assert that "[examples] do not need to appear in pairs, and there is no need to use any method to inform the LLM how they correspond to each other, just randomly shuffle... and train the model like general SFT."

Representation learning extends data augmentation by pulling a model's vector representations for similar inputs (e.g. augmented data) closer together with training (Chen et al., 2020). While primarily used for embedding models that solely map inputs to vector spaces, AgMax by Atienza (2021) extends this idea to regular supervised learning (e.g. SFT). Using paired augmentations of an input, an additional loss term is calculated based on a similarity measure (i.e. "agreement") between the model's output vectors (logits) for this pair. They focus on neural network estimates of mutual information (MI) for agreement, but there exist more common similarity measures like cosine similarity (Balestriero and LeCun, 2022), and lower variance

MI estimates like InfoNCE (van den Oord et al., 2018). More importantly, they introduced a weight hyperparameter to this additional loss, but effects were only investigated for a value of 1.0.

While mostly used in computer vision, there are methods that make use of representation learning in language. SimCSE by Gao et al. (Gao et al., 2021) adapts methods for contrastive learning from vision to train sentence-embedding models. SumCSE by Thirukovalluru et al. (Thirukovalluru et al., 2024) elaborates on this by using summaries as a form of data augmentation. However, these methods are mostly, if not entirely, limited to models for sentence embeddings, and not natural language generation.

We bring attention to the gap in knowledge for using representation learning to train efficient thinking LLMs. Our study aligns C3oT's methods more closely to how CoT SFT is done today with general-purpose CoT training datasets (Team Olmo: Allyson Ettinger et al., 2025). We also investigate its effects on newer-generation models and less saturated benchmarks. Most importantly, we extend AgMax's methods to the language domain using C3oT's data augmentation. We adopt cosine similarity as our agreement function, and investigate the effects of AgMax across weights spanning multiple orders of magnitude.

F Ablations

The following are additional experiments aimed to answer questions on the contribution of different parts of our intervention. We focus on Phi-4-mini-instruct with AMC23 as these setups show substantial improvement with AgShort on our main experimental results.

Setups		Acc. (%) ↑	Len. (tok.) ↓	Eff. (S_{ock})
C3oT		22.5	639	14.4
Pair-only	+AgMax-0.0	(+4.0) 26.5	(-78) 561	(+4.6) 19.0
Full-Answer & Cosine	+AgMax-0.1	(+8.0) 30.5	(-42) 597	(+8.3) 22.7
	+AgMax-1.0	(+8.5) 31.0	(-97) 542	(+9.2) 23.7
	+AgMax-10.0	(+3.0) 25.5	(-90) 549	(+3.7) 18.1
First-Token	+AgMax-0.1	(+6.5) 29.0	(-132) 507	(+7.5) 22.0
	+AgMax-1.0	(0.0) 22.5	(-130) 509	(+1.0) 15.4
	+AgMax-10.0	(+1.5) 24.0	(-162) 477	(+2.8) 17.2
InfoNCE	+AgMax-0.1	(+3.0) 25.5	(-187) 452	(+4.5) 19.0
	+AgMax-1.0	(+2.0) 24.5	(-89) 550	(+2.7) 17.1
	+AgMax-10.0	(-5.5) 17.0	(-100) 539	(-4.8) 9.7
MSE	+AgMax-0.1	(+2.0) 24.5	(-242) 397	(+4.1) 18.5
	+AgMax-1.0	(-5.5) 17.0	(-217) 422	(-3.7) 10.7
	+AgMax-10.0	(-10.5) 12.0	(+94) 733	(–11.1) 3.4

Table 2: AMC23 scores on Phi-4-mini-instruct trained on our ablation setups.

Does paired sampling on its own improve data augmentation? To answer this question, we train a model with an AgMax weight of 0.0. The only difference of this model from one trained with plain C3oT is, therefore, that it saw both long and short CoT versions of samples together during training. Table 2 shows that this on its own already improves accuracy and efficiency by ∼4%. This suggests that data ordering for this type of augmentation on CoT examples is non-trivial, and simple random sampling (C3oT) is sub-par.

Can agreement be done on only the first answer token? In the original AgMax, agreement is only done on one pair of logits for each corresponding augmented samples. This can technically be done in our method as well using only the starting token of the answer portion: `.answer`. However, we find this extremely wasteful of the signal present in the rest of the many succeeding answer logits. Table 2 shows that this first-token agreement can closely match the gains of full-answer agreement at AgMax of 0.1 and 10.0, but severely underperforms at a weight of 1.0 (∼8.22% worse efficiency). This suggests that this approach may be less stable / robust compared to full-answer agreement.

How do other agreement functions perform? We choose cosine similarity as a simpler and more common similarity metric for vectors instead of a neural network estimate of MI for AgMax. However, functions like InfoNCE provide a low variance lower bound estimate for MI. And mean squared error (MSE) is an arguably simpler metric also investigated in the original AgMax. Table 2 shows that cosine similarity outperforms both of these metrics by a substantial margin in all weights. InfoNCE provides improvement over C3oT until the weight of 10.0, while MSE underperforms as early as 1.0. This suggests that these functions do not only consistently underperform compared to cosine similarity, but they are less robust to changes in AgMax's weight hyperparameter.