Compressed Chain of Thought: Efficient Reasoning through Dense Representations

Jeffrey Cheng [1] **Benjamin Van Durme** [1]

Abstract

Chain-of-thought (CoT) decoding enables language models to improve reasoning performance at the cost of high generation latency in decoding. Recent proposals have explored variants of *contemplation tokens*, a term we introduce that refers to special tokens used during inference to allow for extra computation. Prior work has considered fixed-length sequences drawn from a *discrete* set of embeddings as contemplation tokens. Here we propose Compressed Chain-of-Thought (CCoT), a framework to generate *contentful* and *continuous* contemplation tokens of variable sequence length. The generated contemplation tokens are compressed representations of explicit reasoning chains, and our method can be applied to off-the-shelf decoder language models. Through experiments, we illustrate how CCoT enables additional reasoning over dense contentful representations to achieve corresponding improvements in accuracy. Moreover, the reasoning improvements can be adaptively modified on demand by controlling the number of contemplation tokens generated.

1. Introduction

Chain-of-Thought (CoT) refers to the Large Language Model (LLM) technique in which the model simulates the process of thinking out loud by decomposing a complex question into parts and sequentially reasoning through each step. This behavior can be induced by finetuning on a dataset or human feedback (Liu et al., 2023; Puerto et al., 2024), demonstrating through ICL (Wei et al., 2023), or by providing tuned model instructions (Kojima et al., 2023). While CoT improves the reasoning capabilities of LLMs on a variety of tasks, the improvements come at the cost of a high generation latency. For instance, GPT-4o takes 21.37 seconds to generate a response to the question shown in Figure 1 with CoT prompting, whereas it can answer the same question without CoT prompting in 2.81 seconds, achieving the same answer with an almost 10x speedup.

Past work has utilized what we term *contemplation tokens* as an alternative to explicit CoT reasoning traces (Pfau et al., 2024; Goyal et al., 2024). These are additional tokens used to introduce online memory, allowing for additional computations during inference. Instead of generating a reasoning chain entirely of explicit language tokens, the model conditions on a shorter sequence of contemplation tokens (Section 2). Contemplation tokens can either be *contentful*, grounded in semantically meaningful text, or *noncontentful*. There are many lines of prior work involving *noncontentful* contemplation tokens drawn from a set of *discrete* tokens; this paper introduces *contentful* contemplation tokens that represent reasoning chains performed in *continuous* space.

Our framework, called Compressed Chain of Thought (CCoT), generates contemplation tokens which are compressed representations of language-based reasoning chains. These contemplation tokens are trained through teacher forcing with respect to the gold hidden states corresponding to full reasoning traces. Our framework can be adapted to pretrained LLMs through LoRA finetuning. Moreover, the variable compression ratio during training allows for need-based adjustments to the performance-efficiency tradeoff by controlling the number of tokens generated during inference.

The contributions of this paper are as follows:

1. We finetune pretrained decoder-only LLMs with our new CCOT framework and empirically evaluate their performance and throughput on GSM8K;
2. We establish our framework in context of related work in filler tokens and CoT distillation in terms of performance and efficiency;
3. We extend theoretical results and demonstrate the computational capacity of CCOT contemplations tokens.

2. Related Work

Distillation of Knowledge Chains There has been work in distilling the computations done explicitly when decoding the reasoning chains into computation of the hidden states

[1]Department of Computer Science, Johns Hopkins University, Baltimore, US. Correspondence to: Jeffrey Cheng <jcheng71@jh.edu>.

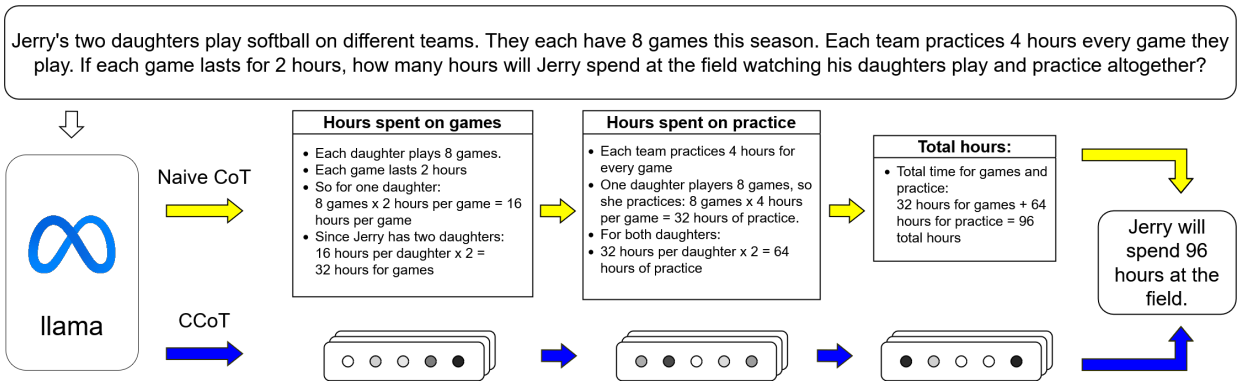

Figure 1. Two approaches to step by step reasoning. Chain of Thought (CoT) prompting reasons via discrete language tokens, leading to long sequences that incur significant generation costs. In contrast Compressed Chain of Thought (CCOT) elicits reasoning with a short sequence of continuous embeddings, allowing for much greater throughput.

of the answer (Deng et al., 2023; 2024). Contemporaneous work distills reasoning paths into continuous latent tokens (Hao et al., 2024). Our method differs in that the contemplation tokens we generate are grounded in text rather than only used as a signal to decode from. This is a critical distinction: our grounding offers the future potential for decoding the reasoning chain from the compressed representations, allowing for post-hoc human inspection of the LLM's reasoning. Moreover, our method successfully adapts a much larger model (7B compared to 1.5B) using a fraction of data (≈ 9000 instances in GSM8K compared to ≈ 400000 instances in an unreleased augmented GSM8K). This suggests that our method can scale better to larger models and is more data efficient.

Filler (Pause) Tokens Many previous methods have considered decoding *contemplation tokens* to provide an LLM with more compute during inference time. These tokens have gone by many names, such as pause tokens (Goyal et al., 2024), memory tokens (Burtsev et al., 2021), filler tokens (Pfau et al., 2024), and thinking tokens (Herel & Mikolov, 2024). These works mainly focus on *noncontentful* contemplation tokens, whose main advantage is their ability to be decoded in parallel, providing the model with a greater computational width without the need to autoregressively decode.

They have been shown to increase the theoretical computational ability of Transformer LLMs (Pfau et al., 2024), but cannot simply be naively applied to induce reasoning gains (Lanham et al., 2023). However, through careful pretraining and finetuning, pause tokens have been shown to improve reasoning in both RNNs (Herel & Mikolov, 2024) and Transformers (Goyal et al., 2024). In contrast, the contemplation tokens generated by CCOT are contentful as they are compressed representations of reasoning chains. Moreover, they are decoded autoregressively resulting in a greater computational depth as well as width.

Contextual Compression Transformer LLMs are the de facto standard architecture for modern NLP applications. However, due to the quadratic complexity of its self-attention mechanism, these LLMs are inefficient in tasks with long contexts. Many techniques have been proposed to alleviate this issue, including memory slots (Ge et al., 2024), dynamic compression into nuggets (Qin et al., 2024), and low level cache encoding (Liu et al., 2024). While most techniques rely on access to the intermediate hidden states of LLMs, there has also been work done in the context of API-only LLMs (Jiang et al., 2023). Overall, most of the work in contextual compression deals with efficient compression of known context in order to improve generation latency. The compressed context can then be used in downstream tasks such as retrieval augmented generation or summarization.

The area of context compression is orthogonal to *comptemplation tokens*. The memory slots of (Ge et al., 2024) and the nuggets from (Qin et al., 2024) encode contentful representations of *known* context, but they are only attended to and never generated during inference. While our work focuses on contentful representations of text, there are two crucial differences: our compressed representations are autoregressively *decoded* during inference and they encode content that is a priori *unknown*.

Chain of Thought Chain-of-thought (Wei et al., 2023) was introduced as a prompting method leveraging in-context learning (ICL) using hand crafted demonstrations. Kojima et al. (2023) showed similar behavior could be elicited in a zero-shot context by instructing a model to "think step-by-step." There have been a variety of innovations to CoT, improving on its efficiency and performance.

In terms of efficiency, novel techniques include getting an LLM to generate steps in parallel from a generated tem-

Method	Contentful	Format	Inference	Additional Notes
Chain of Thought (Wei et al., 2023)	Yes	Discrete	Variable-length; Autoregressively	Best performing method across reasoning tasks; requires no finetuning; inefficient due to unconstrained sequence length.
Filler Tokens (Pfau et al., 2024)	No	Discrete	Fixed-length; In parallel	Explicit example of problems only solvable with contemplation tokens.
Pause Tokens (Goyal et al., 2024)	No	Discrete	Fixed-length; In parallel	Best gains seen when contemplation tokens are added during pretraining stage.
COCONUT (Hao et al., 2024)	Yes	Continuous	Fixed-length; Autoregressively	Trains contemplation tokens by inserting them after removing reasoning steps.
CCOT (**Ours**)	Yes	Continuous	Variable-length; Autoregressively	Trains contemplation tokens to approximate compressed reasoning chains.

Table 1. A comparison of different methods to generate *contemplation tokens* in order to introduce extra computation into models during inference. We characterize several aspects of the tokens: (1) contentful, the tokens are either intrinsically contentful or approximate/are distilled from contentful text; (2) format, whether the tokens are drawn from a discrete set of embeddings or are drawn from continuous space; and (3) inference, how the tokens are generated during inference. Any additional notes for each method are included as well.

plate (Ning et al., 2024) and generating reasoning chains in parallel using Jacobi decoding (Kou et al., 2024; Zhang et al., 2024). In terms of performance, techniques include generating multiple reasoning paths (Yao et al., 2023), and finetuning on human feedback on generated chains (Liu et al., 2023; Puerto et al., 2024). Our method differs from prior work in improving the efficiency of CoT as it is not prompt-based and does not rely on Jacobi decoding.

3. Contemplation Tokens

3.1. Preliminaries and Notation

We first give a brief overview of a causal decoder-only language model, equipped with standard Transformer blocks (Vaswani et al., 2023). Let V be the vocabulary and $w_{1:n}$ be an input sequence, $w_i \in V$. Let d be the hidden dimension, L be the number of layers, and θ be the parameters of the model. The sequence is first passed through an embedding layer, resulting in a vector $w^0_{1:n}$ where each $w^0_i \in \mathbb{R}^d$. The entire vector $w^0_{1:n} \in \mathbb{R}^{n\times d}$ is then passed through a series of Transformer blocks, $T^i : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. We denote the output of each T^i as the *hidden states*. The output of the final Transformer block, $w^L_{1:n} \in \mathbb{R}^{n\times d}$, is then passed through the language model head to generate a distribution $p_{1:n}, p_i \in \mathbb{R}^{|V|}$, from which the next token is sampled.

$$
\begin{aligned}
w^0_{1:n} &= \textsc{embed}_\theta(w_{1:n}) && \triangleright \textit{embedding layer} \\
w^\ell_{1:n} &= \textsc{attn}^{\ell-1}_\theta(w^{\ell-1}_{1:n}) && \triangleright \textit{transformer blocks} \\
p_{1:n} &= \textsc{head}_\theta(w^L_{1:n}) && \triangleright \textit{pass through lm head} \\
p(w_{n+1} \mid w_{1:n}) &\sim p_n && \triangleright \textit{sample next token}
\end{aligned}
$$

Notation-wise, any lowercase letter will refer to a *token*, lying in V. Any lowercase letter with superscripts will refer to the hidden state after passing through the corresponding layer, lying in $\mathbb{R}^d$. Any subscripts refers to a sequence. We will often omit superscripts and instead refer to embeddings with bars ($\bar{}$) and the entire hidden state with hats ($\hat{}$). Under this notation, we instead have $\textsc{embed}(w_{1:n}) = \bar{w}_{1:n}$, and with slight abuse of notation, $\textsc{attn}(\bar{w}_{1:n}) = \hat{w}_{1:n}$.

There are also instances where hidden states of an input are computed under two different sets of weights. Suppose we have two sequences of embeddings $\bar{w}$, $\bar{x}$, and we want to compute the hidden states $\hat{w}$ under weights θ and compute the hidden states of $\hat{x}$ under ψ, but crucially conditioned on $\hat{w}$. In this case, we will write $\textsc{attn}_{\theta,\psi}([\bar{w};\bar{x}]) = [\hat{w};\hat{x}]$ where semicolons indicate vector concatenation.

3.2. Motivation

In question-answer settings, the input $w_{1:n}$ is a query, and the answer $w_{n+1:n+o} = a_{1:o}$ is generated autoregressively as described above. However as seen in the above description of forward passes through Transformer models, the amount of computations for each query is directly proportional to the query length n. As such, we can introduce more computations to the model by attending to an a set of *contemplation tokens*, defined to be any additional tokens generated during inference used to introduce addition memory allowing for additional computations during inference. Rather than solely attending to a query $q = w_{1:n}$, we first can generate a set of contemplation tokens $t = t_{1:m}$ and attend to $[q;t]$ in order to decode a better answer. We emphasize that contemplation tokens are not a novel idea, but a term introduced to unify the many names given to this

concept (Section 2).

We define the contemplation tokens t to be *contentful* if either the tokens themselves are semantically contentful or the hidden states corresponding to the contemplation tokens are derived from semantically contentful tokens. We define contemplation tokens that do not fulfill either of these conditions to be *noncontentful*. An example of contentful contemplation tokens are the reasoning chains in chain of thought (Wei et al., 2023); they describe the model's reasoning, fulfilling the first condition of being semantically meaningful. On the other hand, an example of noncontentful contemplation tokens are filler tokens (Pfau et al., 2024), as they are simply period characters and their hidden states are trained without any signal from semantically contentful hidden states.

Chain of thought turns out to be the only prior method involving contentful contemplation tokens. The performance gains from utilizing chain of thought are clear; however these benefits are offset by the high generation latency. Suppose the input query consists of n tokens and its corresponding reasoning chain consists of m tokens. As each of the tokens in the reasoning chain need to be autoregressively decoded, the generation of the reasoning chain incurs the cost of m extra passes through the model. Moreover when decoding the answer, the model has to attend to the additional m tokens, resulting in $O(m^2)$ more computations when passing through each attention module. As reasoning chains are often many times longer than the query, the amount of extra computations increases dramatically.[1]

3.3. Compressing Reasoning Chains

Prior work showed that *noncontentful* contemplation tokens only improved reasoning when the task was computationally bottlenecked (Pfau et al., 2024) or when the tokens were introduced during pretraining (Goyal et al., 2024). We instead aim to utilize *contentful* contemplation tokens as we believe they would be more applicable to a wider set of tasks. To generate contentful contemplation tokens, we take inspiration from an empirical observation of CoT decoding.

Suppose we have an input query $w_{1:n}$ and its corresponding reasoning chain $t_{1:m}$. We compute the hidden states of concatenated input as $x = [\hat{w}_{1:n}; \hat{t}_{1:m}]$. Decoding an answer conditioned on the hidden states x is equivalent to prompting a language model with the query and chain of thought. Consider taking a subset of $\hat{t}_{1:m}$ along the sequence length axis, denoted as $z_{1:k}$ for some $k << m$. Specifically, for each $1 \leq i \leq k$, there exists some $1 \leq j \leq m$ such that $z_i = \hat{t}_j$ at each layer. We observe that training an adapter to decode conditioning on this shortened $[x_{1:n}; z_{1:k}]$ results in lossless performance on downstream tasks.

Given a query q, the naive method utilizing this observation would be to autoregressively generate the reasoning chain t, select some learned subset of the encoded hidden states z, and train an adapter to decode from the query and subset of hidden states. While this method results in a shorter input sequence when generating the answer and thus reduces the attention computations when decoding the answer, it would still incur the linear cost in generating the reasoning chain. We instead propose learning a module to generate the compressed representations z directly. We denote this module as CCOT, short for **c**ompressed **c**hain **o**f **t**hought, as the contemplation tokens it generates are compressed representations of reasoning chains instead of the full chain.

4. Approach

Assume we have a pretrained causal decoder-only language model LM, parameterized by weights θ. We wish to train two modules, CCOT and DECODE, respectively parameterized by weights φ and ψ. At a high level given a query, CCOT_φ is responsible for the generation of contemplation tokens. DECODE_ψ is responsible for decoding the answer conditioned on the initial query and contemplation tokens.

Consider a training instance consisting of a query, full reasoning chain and answer, denoted as $w_{1:n}, t_{1:m}$ and $a_{1:o}$, respectively. Assume some fixed compression ratio $0 < r < 1$ and let $k = \lceil r \cdot m \rceil$. This compression ratio controls how much the reasoning chains are compressed; $r = 1$ corresponds to finetuning on the full reasoning chain while a $r = 0$ corresponds to finetuning on just the answer. φ and ψ are fine-tuned successively, each initialized from θ.

4.1. Finetuning CCOT_φ

The goal of CCOT_φ is to generate contemplation tokens. Under CCOT, these tokens are a compressed representation of a full reasoning chain, equivalent to a size k subset of the hidden states $\hat{t}_{1:m}$ produced by LM_θ. Since processing all of t and then performing a subset selection still incurs the linear cost of generating all m tokens, CCOT_φ is thus trained to **approximate a subset of precomputed hidden states**.

To achieve this, we first precompute the hidden states of the concatenated input. We next use a checkpoint of a scorer used to perform a similar subset selection from Qin et al. (2024) in order to perform the subset selection of the hidden states. This scorer is simply a linear layer that takes the embeddings from a predetermined layer T as input, and returns the indices of the selected subset. We discuss other

[1] The average reasoning chain in GSM is 1.5 times longer than their corresponding query. Reasoning chains provided by GPT o1 are hundred of times longer than their query.

methods of subset selection in Section 5.2.

$$[\bar{w}_{1:n}; \bar{t}_{1:m}; \bar{a}_{1:o}] = \text{EMBED}_\theta([w_{1:n}; t_{1:m}; a_{1:o}])$$
$$[\hat{w}_{1:n}; \hat{t}_{1:m}; \hat{a}_{1:o}] = \text{ATTN}_\theta([\bar{x}_{1:n}; \bar{t}_{1:m}; \bar{a}_{1:o}])$$
$$I = \text{SCORER}(\hat{t}^T_{1:m})$$

We have that $|I| = k$, and we can index the hidden states $z_{1:k} = \hat{w}_I$ to serve as the gold labels. We aim to generate k contemplation tokens $\hat{z}_{1:k}$ conditioned on $w_{1:n}$ under φ to approximate the labels, but is not immediately clear what inputs we should use to generate the contemplation tokens.

A reasonable choice is to use the embeddings of the tokens corresponding to the selected indices, $\hat{w}_I$. This choice would make the hidden state approximation easier due to skip connections in the attention layer: $\hat{w}_I$ are the exact inputs used to compute the hidden states in the noncompressed case. However, the selected tokens are usually punctuation tokens and articles. This choice would require predicting a random sequence of semantically empty tokens when autoregressively decoding as we pass the last layer embeddings $\hat{z}^L_i$ through the language model head. Another option would be to learn a single embedding as input to generate each hidden state, but this choice removes the additional computational depth induced by autoregressive decoding.

We instead take inspiration from reasoning over continuous space and use the intermediate hidden layers of the previous contemplation token as input to the next token. Formally, the inputs to generate the contemplation tokens $\hat{z}_{1:k}$ are the embeddings of $z^l_{0:k-1}$ at some fixed layer l where z_0 represents the hidden state of the last token of the query.

This choice is quite natural as it generalizes the naive autoregressive decoding strategy (Section 4.3). We train the parameters of φ layer by layer with the following loss:

$$\text{LOSS}_\varphi(z^l_i, \hat{z}^l_i) = \frac{1}{k}\sum_{i=1}^{k} \frac{1}{\sigma^2(z^l_i)} \text{MSE}(z^l_i, \hat{z}^l_i)$$

where $\sigma^2(z)$ denotes the variance of z and MSE denotes the usual mean squared error between two vectors. We use a scaled mean squared error in order to normalize hidden states with average L^1 norms. These norms differ drastically between different layers within the same model, so the scaled loss allows us to keep a consistent learning rate.

To train the ith layer, we pass in the inputs described above and compute forward passes through i Transformer layers, crucially only updating the parameters corresponding to the ith layer. When training subsequent layers, the parameters corresponding to the ith layer are frozen. This provides a natural segmentation to the approximation task, and we found this improved the generated contemplation tokens.

4.2. Finetuning DECODE_ψ

We assume a trained module CCOT_φ. Compressed reasoning chains are out of distribution for θ, so we need a separate module in order to effectively condition on the generated contemplation tokens. We train DECODE_ψ to **decode the answer from the query and contemplation tokens**.

To do this, we first encode the hidden states of the query and autoregressively generate contemplation tokens $z^*_{1:k}$. Con-

Algorithm 1 Chain of Thought inference

Require: Query w, parameters θ
$\bar{w} \leftarrow \text{EMBED}_\theta(w)$ ▷ *embed query*
$\hat{w} \leftarrow \text{ATTN}_\theta(\bar{w})$ ▷ *compute hidden states*
$z \leftarrow [\langle COT \rangle]$
while $z_{-1} \neq \langle ANS \rangle$ **do**
  $[\hat{w}; \hat{z}] \leftarrow \text{ATTN}_\theta([\bar{w}; \text{EMBED}_\theta(z)])$
  $x \sim \text{HEAD}_\theta(\hat{z}^L_{-1})$
  $z \leftarrow [z; x]$
end while
$a \leftarrow [\langle ANS \rangle]$
while $a_{-1} \neq \langle EOS \rangle$ **do**
  $[\hat{w}; \hat{z}; \hat{a}] \leftarrow \text{ATTN}_\theta([\bar{w}_{1:n}; \text{EMBED}_\theta([z; a])])$
  $x \sim \text{HEAD}_\theta(\hat{a}^L_{-1})$ ▷ *sample answer token*
  $a \leftarrow [a; x]$
end while
return a

Algorithm 2 CCOT inference

Require: Query w, parameters θ, φ, ψ, autoregressive layer l
$\bar{w} \leftarrow \text{EMBED}_\theta(w)$ ▷ *embed query*
$\hat{w} \leftarrow \text{ATTN}_\theta(\bar{w})$ ▷ *compute hidden states*
$z \leftarrow [\hat{w}^l_{-1}]$
while $\text{END}_\psi(\hat{z}^L)$ is $False$ **do**
  $[\hat{w}; \hat{z}] \leftarrow \text{ATTN}_{\theta,\varphi}([\bar{w}; z])$ ▷ *gen. cont. token*
  $z \leftarrow [z; \hat{z}^l_{-1}]$ ▷ *append cont. token*
end while
$a \leftarrow [\langle ANS \rangle]$
while $a_{-1} \neq \langle EOS \rangle$ **do**
  $[\hat{w}; \hat{z}; \hat{a}] \leftarrow \text{ATTN}_{\theta,\varphi,\psi}([\bar{w}_{1:n}; z; \text{EMBED}_\theta(a)])$
  $x \sim \text{HEAD}_\psi(\hat{a}^L_{-1})$ ▷ *sample answer token*
  $a \leftarrow [a : x]$
end while
return a

Figure 2. Two algorithms for inference with contemplation tokens. Algorithm 1 describes the usual chain of thought decoding while Algorithm 2 describes our method, obtained by replacing the yellow text with the cyan text. While CoT decoding decodes contemplation tokens by passing the LM head across the final hidden state, we use the hidden state at the lth layer directly.

trasting the training of CCOT_φ, we perform this generation **autoregressively** rather than using the precomputed embeddings $z^l_{0:k-1}$ described in Section 4.1 We start by passing in z^l_0 and compute the hidden states $\hat{z}_1$. We then autoregressively take $\hat{z}^l_1$ as the next input to generate $\hat{z}_2$, until an entire sequence $\hat{z}_{1:k}$ is generated. Then, conditioning on the query and contemplation tokens, we pass in the answer tokens $a_{1:o}$ and compute the next-token distributions $p_{1:o}$.

We finetune ψ with the usual cross-entropy loss given the computed distributions where the probabilities of the next token a_i are drawn from the distribution p_{i-1}.

$$\text{LOSS}_\psi(a_{1:o}) = -\sum_{i=2}^{o} \log p(a_i \mid a_{1:i-1}) \sim p_{i-1}$$

The tokens of a are conditioned on the contemplation tokens $\hat{z}$ generated under φ. By unfreezing the parameters φ when finetuning the parameters ψ using LOSS_ψ, we note that the parameters φ receive signal from the downstream task.

Empirically, we find that this signal is not entirely useful – downstream performance decreased if all the parameters φ are unfrozen. We hypothesize that updating the parameters corresponding to earlier layers affects the autoregressive generation of the contemplation tokens. As such, we find that unfreezing the parameters corresponding to layers *after* the autoregressive layer l ends up improving performance.

k will not be known during test time, so we additionally train a binary classifier END_ψ that takes the final layer of generated hidden states $\hat{z}^L_i$ as input and either predicts whether another contemplation token token should be generated. We stop generating contemplation tokens after h tokens. We set $h = 200r$, which would only prematurely terminate less than 3% of the long tailed distribution of reasoning chains.

4.3. Inference

Assume we have a pretrained causal decoder-only language model parameterized by weights θ. Additionally, assume trained modules CCOT_φ, DECODE_ψ and the end predictor END_ψ. Given a query w, we describe inference in Algorithm 2. We remark that our method to generate contemplation tokens is quite natural; Algorithm 1 describes the usual chain of thought inference and the differences are marked, cyan for our method and Yellow for naive CoT decoding.

The most crucial difference is that when CCOT generates contemplation tokens (lines 4-7), it uses lth layer of the last token's hidden state as a *continuous* next input. In contrast when CoT generates contemplation tokens, it uses the final Lth layer to do the usual autoregressive decoding described in Section 3.1, passing to a *discrete* set of tokens. Moreover, if m is the average length of reasoning chains under θ, CCOT will generate on average only $k = \lceil r \times m \rceil$ contemplation tokens, whereas CoT will generate on average all m tokens.

4.4. Implementation Details

We use LORA (Hu et al., 2021) in order to finetune φ and ψ with ranks of 128 and 64, respectively. When generating the gold hidden states, we pass the $T = 3$ layer to perform our subset selection and the $l = 15$ as inputs. We also take the hidden state at the lth layer to do autoregressive generation of contemplation tokens when finetuning ψ and during inference. We use the decoder-only Transformer architecture of LLAMA for our experiments, taking the LLAMA2-7B-CHAT checkpoint (Touvron et al., 2023) as our base model.

5. Experiments

5.1. Experimental Setups

We evaluate our CCOT framework on the reasoning dataset GSM8K (Cobbe et al., 2021). For the reasoning chains required to train both modules, we use the chains of thought provided with the dataset. We remove all calculator annotations present in the reasoning chain, only keeping the natural language reasoning. We finetune φ with precomputed gold states with two compression ratios, $r = [0.05, 0.10]$. We emphasize that the choice of r is a training time decision, CCOT_φ approximates the hidden states under the fixed compression ratio r.

We compare our results to two baselines of $r = [0.0, 1.0]$. These compression ratios are the two extreme values of the compression spectrum we introduce, corresponding to the cases of no reasoning chain and full reasoning chain. We finetune the model with the usual cross entropy loss on the dataset; For $r = 0.0$, the model directly outputs the answer without generating any contemplation tokens. For $r = 1.0$, the model generates the explicit reasoning chain as its contemplation tokens during inference.

Additionally, we compare to PAUSE, a method derived from Goyal et al. (2024). We finetune the model with no reasoning chains, but for a given ratio r, append $k = \lceil r \times m \rceil$ contemplation tokens between the query and answer where m is the length of the reasoning chain. We learn the input embedding of the special token, chosen to be $\langle pause \rangle$. These pause tokens are added to provide the model with an enhanced computational width (See Section 6.2 for further discussion). We evaluate with the same compression ratios $r = [0.05, 0.10]$ to measure the effect of the tokens.

5.2. Results and Discussion

We provide our main results in Table 2. Accuracy refers to the exact match accuracy obtained on the test set with no in-context examples. Decode time refers to the average time to generate an answer to a test set query, measured in

Format	$1/r$	Acc. (EM)	Decode Time
CCOT	∞	0.089	0.33
CCOT	20x	0.151	0.49
CCOT	10x	0.179	0.78
CCOT	1x	0.315	8.10
PAUSE	20x	0.092	0.35
PAUSE	10x	0.099	0.37

Table 2. Accuracy and decode time on GSM8K (Cobbe et al., 2021) contrasting our method, CCOT, and PAUSE (Goyal et al., 2024) each equipped with two different compression ratios against baselines of no compression (full reasoning chains) and infinite compression (no contemplation tokens). Higher accuracy indicates better performance, while lower decode time indicates better efficiency.

seconds by wall clock time on a single Nvidia A100 GPU.

With a compression ratio of $r = 0.10$, we see a 9 point improvement over the baseline with no contemplation tokens. This accuracy gain is achieved with an only 0.4 second increase in generation time. If we reduce r to 0.05, we still see a sizable 6 point improvement over the baseline, with a generation time increase of only around 0.15 seconds. In contrast, even though the contemplation tokens generated by PAUSE could be decoded faster, they were only able to nominally improve performance. We hypothesize that even though these tokens provide the model with additional computations, reasoning datasets like GSM8K require more sequential computations over parallel ones. Ultimately, our results show equipping a model with dense, contentful contemplation tokens produced by CCOT allows the model to reason better than if it had no contemplation tokens, or used a discrete set of noncontentful ones.

6. Further Discussion

6.1. Hyperparameter Choices

Varying r As r controls how many contemplation tokens are generated, it makes sense that increasing r would increase both accuracy and decode time. However, we found that accuracy plateaus after a certain threshold, about $r = 0.2$. We hypothesize that this occurs because successive contemplation tokens are autoregressively decoded using the hidden state at the l layer, which propagates an approximation to the next contemplation token generation. We suspect the noise from the approximation errors eventually outweighs the signal provided by the contemplation tokens.

Varying l We find that the choice of l is important – we were unable to learn good weights for φ when l was set close to either 0 or the last layer L. We hypothesize that hidden states at earlier layers (small l) still incorporate a lot of localized information about the token itself while hidden states at later layers (large l) instead incorporate a lot of localized information about the *next* token. As such, we found that $l \approx L/2$ resulted in the best performance; we hypothesize that the hidden states at intermediate layers encode global information it them suitable for autoregressive decoding scheme we use to generate contemplation tokens. We provide results with other layer choices in Appendix A.

Subset selection We used a learned scorer module to perform the subset selection of the gold hidden states to be emulated by φ. In practice, we found that simply taking k evenly spaced tokens resulted in a similar performance. However, we note that a module trained to decode from gold hidden states (in the setup described in Section 3.3) achieves lossless performance compared to decoding from the full reasoning chain, even for small values of r. As such, we hypothesize that it is possible to learn a better scorer to identify a subset of hidden states that is easier to emulate; a better approximation of the gold hidden states could lead to lossless performance while only taking a fraction of the time to decode. The observed performance-efficiency tradeoff also likely occurs because it is easier to approximate sequences with less compression.

6.2. Theoretical Considerations

We explore the enhanced computational expressivity offered by contemplation tokens and crucially identify the advantage of decoding contemplation tokens autoregressively rather than in parallel. We provide a few high level intuitions that are formalized in Appendix B.

Width Suppose we have a Transformer block ATTN and an input $\bar{w}_{1:n}$. Computing $\text{ATTN}(\bar{w}_{1:n})$ results in $O(n)$ parallel operations. If we pass in m additional contemplation tokens in parallel and compute $\text{ATTN}(\bar{w}_{1:n+m})$, we now perform $O(n+m)$ parallel operations. The extra computations matter in tasks when the number of parallel operations required is greater than the input sequence length. This can occur when answering succinctly phrased problems that require many parallel operations: "compute all pairwise sums in the following list" or "select all combinations of dependent logical statements that can be mutually true" Computing pairwise sums of an n element list requires processing $O(n^2)$ parallel computations and computing the validity of all possible logical combinations of n facts requires processing $O(2^n)$ ones. As n grows, introducing contemplation tokens during inference in these *width-bottlenecked* scenarios can allow models to solve additional problems.

Depth Suppose we have a model consisting of L Transformer blocks, and we generate contemplation tokens autoregressively than in parallel. Passing in m additional contemplation tokens still results in the increased $O(n+m)$ parallel

operations, but also resuts in $O(mL)$ sequential operations. These extra computations matter in tasks when the number of sequential operations required is greater than the depth of the model. This can occur in multi-hop question answering tasks or when determining the best move in sequential games such as go and chess. Introducing autoregressively decoded contemplation tokens in these *depth-bottlenecked* scenarios can allow models to solve additional problems.

To collect these observations into a formal theorem, we build from prior work that provides an analysis of the computational power of contemplation tokens decoded in parallel (Goyal et al., 2024). We restate their theorem below:

Theorem 6.1 (From Goyal et al. (2024)). *Assume that the attention module has sufficiently many parameters (K) that is much larger than the number of input tokens (N). Then there are tasks that M independent computations, where $N < M < K$), such that a 2-layer Transformer can implement the task if and only if it uses contemplation tokens.*

Under the same assumptions, autoregressively decoded contemplation tokens can solve a broader class of problems. When the depth of a task D exceeds the number of layers in a model L, the model can only represent L steps out of the required D steps. Our intuition is that contemplation tokens can "save" the intermediate steps, so autoregressively the model's representation of the Lth step as the input to the next token allows for the next forward pass to implement another L steps on top of the saved work. Thus, any task of depth D can be solved with an additional D/L contemplation tokens. We note that in CCOT, we only pass in the representation of the $l \approx L/2$ step, but this doesn't detract from the asymptotic representational capacity; we simply require an additional D/l tokens instead of D/L. We informally state our theorem below, see Theorem B.5.

Theorem 6.2. *Assume the conditions in Theorem 6.1. Then, there are tasks that involve M independent computations of a depth $D > 2$ such that a 2-layer Transformer can implement the task if and only if it autoregressively decodes contemplation tokens.*

7. Conclusion

We propose a new framework, CCOT, to generate contentful and autoregressively decoded contemplation tokens, a term we introduce to unify the terms given to tokens introducing additional computation to a language model. CCOT provides substantial improvements over the baseline as well as methods introduced by prior work. We additionally show how reasoning can be viewed as efficiency-performance tradeoff through the adaptive compression ratio. Overall, our work demonstrates the potential of using contentful contemplation tokens as an alternative to explicit reasoning chains, suggesting a new paradigm of reasoning in continuous space.

References

Burtsev, M. S., Kuratov, Y., Peganov, A., and Sapunov, G. V. Memory transformer, 2021. URL https://arxiv.org/abs/2006.11527.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Deng, Y., Prasad, K., Fernandez, R., Smolensky, P., Chaudhary, V., and Shieber, S. Implicit chain of thought reasoning via knowledge distillation, 2023. URL https://arxiv.org/abs/2311.01460.

Deng, Y., Choi, Y., and Shieber, S. From explicit cot to implicit cot: Learning to internalize cot step by step, 2024. URL https://arxiv.org/abs/2405.14838.

Ge, T., Hu, J., Wang, L., Wang, X., Chen, S.-Q., and Wei, F. In-context autoencoder for context compression in a large language model, 2024. URL https://arxiv.org/abs/2307.06945.

Goyal, S., Ji, Z., Rawat, A. S., Menon, A. K., Kumar, S., and Nagarajan, V. Think before you speak: Training language models with pause tokens, 2024. URL https://arxiv.org/abs/2310.02226.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space, 2024. URL https://arxiv.org/abs/2412.06769.

Herel, D. and Mikolov, T. Thinking tokens for language modeling, 2024. URL https://arxiv.org/abs/2405.08644.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. Llmlingua: Compressing prompts for accelerated inference of large language models, 2023. URL https://arxiv.org/abs/2310.05736.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Kou, S., Hu, L., He, Z., Deng, Z., and Zhang, H. Cllms: Consistency large language models, 2024. URL https://arxiv.org/abs/2403.00835.

Lanham, T., Chen, A., Radhakrishnan, A., Steiner, B., Denison, C., Hernandez, D., Li, D., Durmus, E., Hubinger, E., Kernion, J., Lukošiūtė, K., Nguyen, K., Cheng, N., Joseph, N., Schiefer, N., Rausch, O., Larson, R., McCandlish, S., Kundu, S., Kadavath, S., Yang, S., Henighan, T., Maxwell, T., Telleen-Lawton, T., Hume, T., Hatfield-Dodds, Z., Kaplan, J., Brauner, J., Bowman, S. R., and Perez, E. Measuring faithfulness in chain-of-thought reasoning, 2023. URL https://arxiv.org/abs/2307.13702.

Liu, H., Sferrazza, C., and Abbeel, P. Chain of hindsight aligns language models with feedback, 2023. URL https://arxiv.org/abs/2302.02676.

Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., Maire, M., Hoffmann, H., Holtzman, A., and Jiang, J. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pp. 38–56, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706141. doi: 10.1145/3651890.3672274. URL https://doi.org/10.1145/3651890.3672274.

Ning, X., Lin, Z., Zhou, Z., Wang, Z., Yang, H., and Wang, Y. Skeleton-of-thought: Prompting llms for efficient parallel generation, 2024. URL https://arxiv.org/abs/2307.15337.

Pfau, J., Merrill, W., and Bowman, S. R. Let's think dot by dot: Hidden computation in transformer language models, 2024. URL https://arxiv.org/abs/2404.15758.

Puerto, H., Chubakov, T., Zhu, X., Madabushi, H. T., and Gurevych, I. Fine-tuning with divergent chains of thought boosts reasoning through self-correction in language models, 2024. URL https://arxiv.org/abs/2407.03181.

Qin, G., Rosset, C., Chau, E. C., Rao, N., and Durme, B. V. Dodo: Dynamic contextual compression for decoder-only lms, 2024. URL https://arxiv.org/abs/2310.02409.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Zhang, H., Liu, Z., Zhao, Y., Zheng, J., Zhuang, C., Gu, J., and Chen, G. Fast chain-of-thought: A glance of future from parallel decoding leads to answers faster, 2024. URL https://arxiv.org/abs/2311.08263.

A. Varying the autoregressive layer

Our method CCOT autoregressively generates contemplation tokens by using the hidden state at the lth layer at index i as the input embedding at index $i+1$. We show the results of varying this autoregressive layer l. We have that $l = 0$ corresponds to the embedding layer and $l = L$ corresponds to the final layer prior to passing through the model head.

Autoregressive Layer	Accuracy (EM)
NONE	0.089
3	0.087
15	0.151
31	0.092

Table 3. Accuracy on GSM8K with our method CCOT with a compression ratio of $r = 0.05$ when varying the autoregressive layer l. NONE refers to the baseline where no contemplation tokens are decoded during inference.

B. Further Theoretical Considerations

In this section, we formalize the two insights outlined in Section 6.2. We note that an analysis of the enhanced computation width provided by contemplation tokens decoded in parallel was provided by Goyal et al. (2024). They established a series of assumptions and defined a class of problems involving many parallel operations that a 2-layer Transformer is able to solve only if it leverages contemplation tokens.

We observe that any tasks able to be solved by decoding contemplation tokens in parallel can also be solved by decoding contemplation tokens autoregressively. We thus extend the results from Goyal et al. (2024) by defining a more general set of problems that a 2-layer Transformer is able to solve only if it decodes contemplation tokens autoregressively.

In CCOT, contemplation tokens are decoded autoregressively by using the hidden state at the lth layer as the next input. As we take $l \approx L/2$, we adapt this framework to a 2-layer Transformer by using the only intermediate layer, $l = 1$. We formally introduce the new class of problems and outline the assumptions made by Goyal et al. (2024) below.

Assumption B.1. ***(structure of underlying task)*** *Assume a vocabulary $\mathcal{V}$ and a embedding dimension of d. Let $\circ$ be a genetic 2-ary operator on the embedding space $\mathbb{R}^d$. For a given input length N, define the class of functions $\mathcal{F}_{M,K}$ to be the set of all functions $f : \mathcal{V}^N \to \mathcal{V}$ that require applying computing M different $\circ$ operations of depth K, followed by a generic aggregation function $g : \mathbb{R}^{M\times d} \to \mathcal{V}$.*

Here, we assume that the vocabulary is passed into the embedding space through an embedding layer prior to the Transformer blocks. This embedding layer is given as $h : \mathcal{V} \to \mathbb{R}^d$. We define $\mathcal{F}_{\mathcal{M},\mathcal{K}}$ symbolically as

$$\mathcal{F}_{M,K} = \Big\{ f : \mathcal{V}^N \to \mathcal{V} \;\Big|\; \exists T_j \in \mathcal{P}(\{1,\cdots,N\}) \; |T_j| = K, \forall j \in \{1,\cdots,M\} \; f(v_1,\cdots,v_N) = g(\bigcirc_{i\in T_1} \bar{v}_i, \ldots, \bigcirc_{i\in T_M} \bar{v}_i) \Big\}$$

Previous work only considered the case of $K = 2$ (Goyal et al., 2024), which lends itself well to parallel tasks. This structure extends this case by considering inputs to g that require more sequential computation. Examples of these tasks that require more sequential computations include computing the sums of all triplets in a list of numbers, multi-hop QA tasks, and generally any problem requiring recursion.

The following assumptions are taken from Goyal et al. (2024). Further details can be found in the original paper.

Assumption B.2. ***(information bandwidth limit of Transformer hidden states)*** *We assume that the hidden state corresponding to the ith token at any layer can be represented as (u_i, i) by a mapping $h : \mathbb{R}^d \to \mathcal{V} \times \mathbb{N}$.*

Assumption B.3. ***(representational limits of Transformer operations)*** *Let $v \in \mathbb{R}^{N\times d}$ be a sequence of hidden states. Assume that at every index $i \in \{1,\cdots,N\}$, $\text{ATTN}(v)_i$ can represent two types of functions.*

- The $\circ$ operation on the hidden states of two arbitrary indices j, k. We keep the same assumptions that the self-attention module can select the two indices and the feed-forward module can implement the $\circ$ operation, $\text{ATTN}(v)_i = v_j \circ v_k$.
- The aggregating function g, the Transformer block can represent $\text{ATTN}(v)_i = (g(v_1,\cdots,v_N), i)$.

Assumption B.4. ***(the capacity of the Transformer block is independent of input length)*** *We assume that the self-attention module has at least $2T \log T$ parameters for some $T >> N$ and thus can implement any of the T^T possible index mappings This means that the self-attention module can select up to T pairs of .*

Theorem B.5. *Under the conditions outlined in Assumptions B.1 to B.4, the three following statements are true assuming an input sequence of length N.*

- *Standard inference with a 2-layer Transformer can only represent the function class $\mathcal{F}_{M,2}$ for $M \leq N$.*
- *For any $M \leq T$, a 2-layer Transformer that decodes $M - N$ contemplation tokens in parallel can represent the function class $\mathcal{F}_{M,2}$.*
- *For any $K > 2$, $MK \leq T$, a 2-layer Transformer that decodes $MK - M$ contemplation tokens autoregressively can represent the function class $\mathcal{F}_{M,K}$.*

Proof: To prove the first point, it suffices to show that we can represent the function class $\mathcal{F}_{N,2}$ under standard inference with an input sequence length of N tokens. We demonstrate this via construction, computing N distinct $\circ$ operations in the first Transformer block, and the aggregation in the second Transformer block. In order to represent all possible choices of pairs, we need to have the N representations of each token at the embedding layer. Expressing N representations must use all N indices by Assumption B.2. We use the natural choice of using the ith index to represent the ith token's embedding. By Assumption B.4, we can compute the N distinct $\circ$ operations in the first Transformer block, and aggregate them using the second Transformer block. Thus, we show that we can represent $\mathcal{F}_{N,2}$.

To prove the second point, it suffices to show that we can represent the function class $\mathcal{F}_{N+1,2}$ by appending a singular contemplation token. We know from Assumption B.2 that the second Transformer block can aggregate $N+1$ inputs. Assuming $N+1 \leq T$, the first layer can compute the addition $\circ$ operation by Assumption B.4. This argument follows by finite induction for any $N+i \leq T$.

For the last point, we observe that the autoregressive inputs will "save" an intermediate step which allows it to be conditioned on in the same layer. For instance, to compute $\bar{v}_1 \circ \bar{v}_2 \circ \bar{v}_3$ given the input $v = v_1 v_2 v_3$, we would let the output of the first block be $\text{ATTN}(\bar{v})_3 = \bar{v}_1 \circ \bar{v}_2$. This gets passed autoregressively as the next input, denoted as $\bar{w}$. Then $\text{ATTN}(\bar{v})_4$ can select the index of the new token and the index of the third token to compute $\bar{w} \circ \bar{v}_3 = \bar{v}_1 \circ \bar{v}_2 \circ \bar{v}_3$.

In order to compute a $\circ$ operation of depth K, we need to compute the sequential prefix $\circ$ operations of depth $K-1, \cdots, 2$. This requires a total of $K-1$ extra autoregressively generated contemplation tokens just to compute a single $\circ$ operation of depth K. The worst case scenario is that none of the prefix $\circ$ operations are shared, so autoregressively decoding $M(K-1)$ contemplations will allow us to compute all M $\circ$ operations. We have at most MK total tokens, so given $MK \leq T$, we can compute the desired $\circ$ operations by Assumption B.4. $\square$