Fast Chain-of-Thought: A Glimpse of Future From Jacobi Decoding Leads to Answers Faster

Anonymous ACL submission

Abstract

CoT (chain of thought) is widely used in reasoning tasks with large language models (LLM). In this work, we propose FastCoT, a model-agnostic framework to accelerate inference of CoT tasks without training or modification. FastCoT utilizes parallel decoding and autoregressive decoding simultaneously. Parallel decoding generates multiple approximate tokens in a forward, and autoregressive decoding leverages these preliminary approximate tokens to yield one or more refined tokens. Distinct from conventional approaches that rely solely on precisely generated tokens from autoregressive decoding, FastCoT integrates these approximate yet informative tokens in the final response generation process. The approximate tokens act as **a quick glance** of the future tokens, which could lead to faster generation compared to regular autoregressive decoding. Through extensive experiments, we demonstrate that FastCoT accelerates inference by nearly 20% on wide models, with only a negligible performance drop compared to the regular approach.
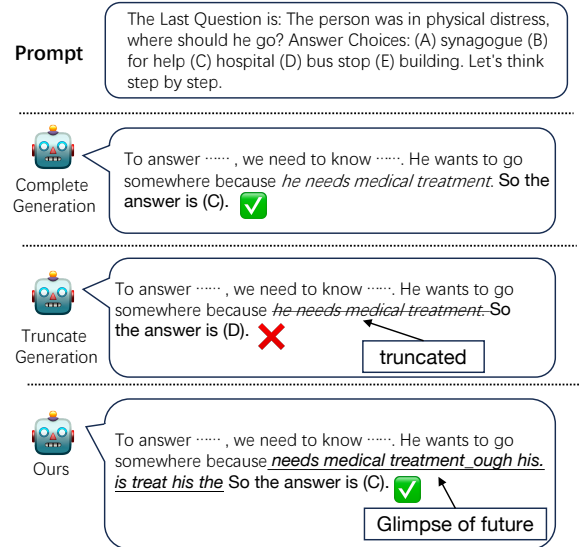
Figure 1: Example of how the glimpse of future work in CoT reasoning task. In the CoT reasoning task, LLM is required to completely generate the complete rationale and finally get the answer based on the rationale. FastCoT argues that partial generation of the complete rationale with a glimpse of future is enough. The example in this figure comes from our experiment results.

1 Introduction

The NLP field has undergone a revolution with the introduction of large language models (LLMs) consisting of tens or hundreds of billions of parameters. These models are pretrained on large-scale corpora using self-supervised training objectives. One notable milestone in this domain is the release of Chat-GPT, a powerful AI chatbot that is developed based on LLMs and has garnered widespread attention from society. Following this breakthrough, several other large-scale language models have been introduced, including Llama (Touvron et al., 2023a,b), PaLM (Chowdhery et al., 2022), and Bloom (Scao et al., 2022), all of which have achieved remarkable success in areas such as language understanding and text generation.

However, several challenging and complex issues still remain unresolved in the field, including arithmetic, commonsense, and symbolic reasoning. In order to enhance the performance of LLMs on various tasks, the Chain of Thoughts (CoT) series of work has been proposed. The concept of CoT prompting was first introduced by (Wei et al., 2022). This series of work aims to improve the performance of LLMs on a wide range of reasoning tasks by guiding the model to generate its own rationales, which are a series of intermediate reasoning steps. Notice that this improvement in performance comes at the cost of generating additional rationales. Since the main focus is often on obtaining the answers, the extra generation of rationales may be seen as a drawback. Furthermore, most state-of-the-art causal transformers are autoregressive models, meaning they can only predict tokens one at a time. This leads to slow inference and

1

may not fully utilize the capabilities of GPUs(Yang et al., 2023).

The nature of autoregressive models imposes significant computational load, which can be limiting for certain real-world applications, especially for industrial applications that involve large amounts of data flow and require high throughput, which would lead to high cost and poor user experience. Therefore, a natural question arises: can a large language model (LLM) benefit from its inner approximate reasoning rationale? It is noteworthy that humans usually think a lot in mind before getting started to write down on paper. Although these vague ideas or thoughts may not be fully connected to form complete sentences, they can still be beneficial for the reasoning task. As shown in Figure 1, we argue that word pieces generated by a fast but inaccurate decoding strategy are informative to generate correct result.

To validate our arguments, we conducted an experiment by corrupting the rationale generated by the LLM, and lead the LLM infer the answer according to the corrupted rationale instead. More details please refer to Section 5.2. A major observation is that, even a randomly sampled oracle rationale can lead to correct answers, indicating that the LLMs are able to make accurate predictions without having access to the complete rationale. Motivated by this, we propose a method called FastCoT, which can utilize approximate rationale to find answer faster. This method combines the exact tokens obtained through any lossless decoding methods with the *approximate* tokens obtained through Jacobi decoding. By doing so, we can obtain the answer to the reasoning task with fewer forward inferences of the LLM. Our contributions can be summarized in three main aspects:

1. We first introduce Jacobi decoding into the reasoning task, such as CoT, and propose to use the by-products of Jacobi decoding, which are approximate tokens, as a glimpse of the future for LLM in the decoding reasoning process. We demonstrate and analyze how Jacobi decoding reduces time overhead in reasoning task scenarios from two aspects: The first aspect comes from Jacobi decoding itself, which can reduce the number of iterations by generating more than one tokens in single forward inference with probability. The second aspect comes from glimpse into the future through approximate tokens, which can help LLM analyze the final answer in advance without autoregressively decoding it.

2. We conduct extensive experiments on LLMs with different scales and datasets, showing speedups of up to 20% in inference time. In the reasoning task scenario, we analyze the time overhead of each part compared to the most common used lossless method, i.e., autoregressive decoding. To the best of our knowledge, this is the first study to accelerate inference of CoT task with inaccurate reasoning steps.

3. Third, we have designed a parallel decoding framework to support a widely range of large language models and will release our framwork to public, with almost no modification to the source code of the language model itself based on the huggingface implementations.

2 Related Work

XoT Prompt Engineering The Chain-of-Thought (CoT) prompting (Wei et al., 2022; Kojima et al., 2022) is proposed to induce large-scale language models to think step-by-step, similar to how humans approach complex questions. A typical few-shot CoT prompt consists of K examples with corresponding rationales and answers for the questions. After that, several methods have been proposed to enhance the performance of CoT prompting across various domains and tasks, including Self-Ask (Press et al., 2022), Self-Consistency (Wang et al., 2022), Self-Verification (Weng et al., 2022), Maieutic Prompting (Jung et al., 2022), Automate-CoT (Shum et al., 2023), MoT (Li and Qiu, 2023), ToT (Long, 2023), and GoT (Besta et al., 2023). Additionally, (Zhang et al., 2022) proposes a clustering-based approach to automatically select questions from different classes to form a CoT Prompt. (Diao et al., 2023) propose using a metric called uncertainty to identify the most uncertain problems in LLM and enhance CoT's performance by providing artificial answers to those questions. These tasks are referred to as XoT. Our work is parallel to these engineering efforts and can be applied to any prompting method related to XoT. In terms of utilizing rationale in XoT Prompting, several works, such as (Magister et al., 2022; Hsieh et al., 2023; Li et al., 2023; Wang et al., 2023), attempt to use CoT-generated rationale to improve the task performance of student models across diverse datasets

2

through rationale distillation. These methods are capable of yielding more accurate results compared to the non-XoT approach. However, they compromise the speed of inference. To the best of our knowledge, we are the first to propose exploiting the concept of approximate rationale to expedite the completion of XoT-series tasks, and our method is orthogonal to the prompt engineering approach.

Accuracy Lossless Decoding Acceleration

Many works have proposed various methods for accelerating the generation of autoregressive causal language models. (Stern et al., 2018) introduced Blockwise Parallel Decoding, utilizing K language-model-dependent decoding heads to predict K more tokens compared to autoregressive decoding. On the other hand, (Kasai et al., 2020) proposed using a deeper Encoder and fewer layers of Decoder to achieve faster speed. (Xia et al., 2022; Leviathan et al., 2023; Chen et al., 2023) suggested employing a faster but less powerful draft model to assist large models in generating multiple tokens in one transformer call. In the field of Machine Translation, (Gu et al., 2017; Huang et al., 2022) introduced Non-Autoregressive Translation (NAT) models to overcome the limitations of autoregressive decoding. However, the application of NAT models is mostly limited to machine translation and related domains, and need careful parameter tuning.

The most closely related work to ours is (Santilli et al., 2023), where they propose the use of Jacobi decoding in translation tasks. However, they do not leverage any approximate tokens generated during Jacobi decoding, and they do not provide an implementation that supports batch computation. To the best of our knowledge, we are the first paper to attempt utilizing these approximate tokens. Compared to the methods mentioned earlier in this section, our framework does not require any additional training on either the base pretrained language model or a new model.

3 Preliminary

3.1 Parallel Jacobi Decoding

For a trained model with parameters θ, a forward inference of the LLM would generate the distribution $p_\theta(y_i|y_{1:i-1}, x)$ for the next token based on the prompt x and the tokens generated so far $y_{1:i-1}$. In the case of a common greedy autoregressive decoding approach without considering any post logits processor, the next token would be selected by choosing the token with the highest probability from the distribution, which can be represented as below.

$$y_i \leftarrow \arg\max p_\theta(y_i|y_{1:i-1}, x) \quad (1)$$

In contrast to autoregressive decoding, Jacobi decoding (Santilli et al., 2023) takes into account additional tokens after the generating position, resulting in a corresponding increase in the number of outputs. To facilitate the description, we refer to the additional tokens considered during forward inference in this context as the *context window*, denoted its size as c. A Jacobi decoding process can be shown as,

$$
\begin{cases}
y_i & \leftarrow \arg\max p_\theta(y_i|x, y_{1:i-1}) \\
\widehat{Y}_{i+1}^t & \leftarrow \arg\max p_\theta(y_{i+1}|x, y_{1:i-1}, \widehat{Y}_i^{t-1}) \\
\widehat{Y}_{i+2}^t & \leftarrow \arg\max p_\theta(y_{i+1}|x, y_{1:i-1}, \widehat{Y}_{i:i+1}^{t-1}) \\
& \vdots \\
\widehat{Y}_{i+c}^t & \leftarrow \arg\max p_\theta(y_{i+c}|x, y_{1:i-1}, \widehat{Y}_{i:i+c-1}^{t-1})
\end{cases}
\quad (2)
$$

For the simplicity of subsequent formulation, we denote one iteration of the Jacobi Decoding as,

$$y_i, \widehat{Y}^t \leftarrow JD(x, y_{<i}, \widehat{Y}^{t-1}, i, c) \quad (3)$$

where i is used to indicate the start position of the Jacobi decoding, and we use y and $\widehat{Y}$ to distinguish the tokens corresponding to the accuracy-lossless decoding part and the Jacobi decoding part. Variable t is used to differentiate the results produced by each iteration. The y^{t-1} means iterative solution of previous iteration, which is also the tth iteration's input to the LLM. We name $\widehat{Y}^t$ as approximate tokens due to they cannot pass the verification and therefore cannot be accepted as text generated by lossless decoding, they are considered as **by-products** of exact parts in previous work. Compared with the previous description of the Jacobi decoding from (Santilli et al., 2023), we specially add the superscript t to highlight the role of approximate tokens generated during the iterative process in our method.

Does context window time consuming? Intuitively, increasing the number of tokens that the model need calculate during inference will lead to an increase in latency. However, we will demonstrate counter-intuitive results through experiments: that is, the context window within a certain limit will not cause an increase of inference latency. We conducted a single inference task to measure the
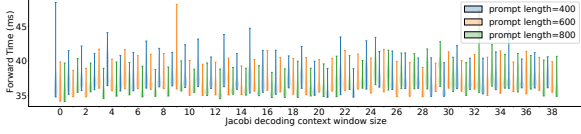
3

Figure 2: Time cost with different context window size c. The time overhead does not increase significantly as the window size increases. Almost no change in time consumption from autoregressive decoding (size=0) to Jacobi decoding with a window size of 38. We apply Llama2-13B model with single Nvidia A100 GPU.

time overhead for a single forward inference under different window sizes, with considering different prompt length. Figure 2 illustrates the results, the forward time almost remains constant with respect to the context length. The primary contributory factor underlying this phenomenon is that, when the decoding length is comparatively limited, the consequent diminutive matrix blocks can not fully leverage the computational capabilities of CUDA and Tensor cores to their fullest extent. Our findings suggest that increasing the window size within a certain range has only a minimal impact on latency for Jacobi decoding.
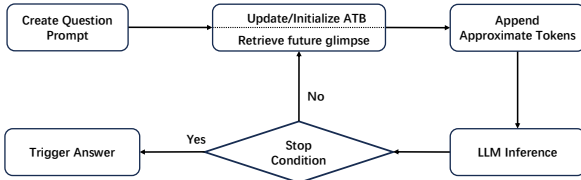
4 Method

Figure 3: Overview of FastCoT. The autoregressive decode get another one exact token, further future's approximate token is revealed by Jacobi decoding.

In our research, we interrogate whether Jacobi decoding by-products can offer a **glimpse of the future**. Current applications and methods in LLMs, such as those employing Jacobi decoding for machine translation ((Santilli et al., 2023)), focus exclusively on precise decoding outcomes, neglecting the potential insights from by-products as they cannot be verified during forward propagation. We thought about whether the by-products part of Jacobi Decoding could help with certain tasks. Considering that by-products mostly consist of tokens related to the final generated text, we call it a glimpse of the future sentence. We would like to emphasize the importance of this *glimpse* into the future, could play an important role in enabling

faster answer generation in XoT-series scenarios.

To begin with, we will briefly introduce the important components of our proposed algorithm. Then, we will introduce how to iterate through approximate tokens through Jacobi decoding.

4.1 Overview of FastCoT

First, in a manner similar to the original CoT methodology, FastCoT would construct a prompt for the question Q that requires reasoning by LLM. Secondly, FastCoT sequentially retrieves approximate tokens from the **Approximate Tokens Buffer** to fill the context window. These tokens are then appended subsequent to the tokenized prompt, thereby establishing an initial input for Jacobi decoding processes. Third, the LLM would conduct the inference call. Subsequent to this step, at least one token precisely matches that which would be produced by autoregressive decoding, thus facilitating the generation of a solution for the subsequent iteration. The newly derived approximate tokens would be used to update the corresponding entries within the Approximate Tokens Buffer. Finally, upon meeting the termination criterion, all generated tokens (including those approximated) are amalgamated with the answer trigger to elicit the final response from the LLM.

4.2 Approximate Tokens Buffer

In contrast to the original CoT rationales elicited by LLMs in standard settings, the rationales employed in FastCoT consist of two distinct components: a precise segment that is identical to the output generated via any accuracy-lossless decoding, and an ambiguous segment comprising *approximate tokens* produced through Jacobi decoding. These approximate tokens, derived from Jacobi decoding, may not precisely match those procured through lossless decoding processes, because of the lack of the verification from language models. In our approach, we use Y to refer the Approximate Tokens Buffer, where $Y = [Y_{0:I}^t, \widehat{Y}_{I:}^t]$. Here $Y_{0:I}^t$ means the exact tokens and $\widehat{Y}_{I:}^t$ represents the approximate part. The vector I contains batch-size number of integers indicating the positions of the first approximate tokens in each instance of the batch. As the decoding process iterates, the approximate tokens in the buffer are gradually transformed into exact tokens. When $t = 0$, FastCoT initialize I as $\mathbf{0}$. We choose to initialize Y with populating tokenized token id sequence of question itself based on empirical results for each question. Compared
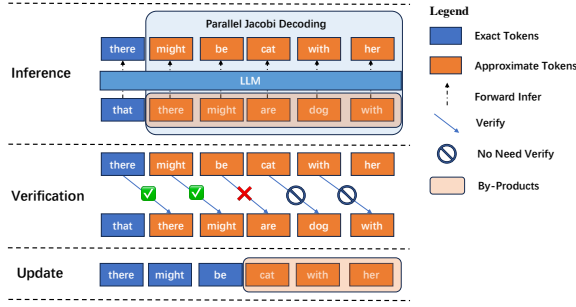
4

Figure 4: Overview of a complete iteration of FastCoT.

with the commonly used initialization method, i.e., initializing using special token [PAD], our method would provide a more diverse initial solution for early stage's Jacobi decoding.

4.3 Approximate Rationale Generation

We will divide the explanation into three parts as show in Figure 4.

FastCoT Inference: During the iterative inference process, lossless decoding and Jacobi decoding are performed in parallel within a single forward, leveraging the causal attention mask used in causal language models. The expression is as follow:

$$y_I, \widehat{Y}_{I+1:I+c+1}^{t+1} \leftarrow PD(x, \mathbf{y}_{<I}, \widehat{Y}_{I:I+c}^t, I, c) \quad (4)$$

Verification: In contrast to autoregressive decoding, where only one exact token is generated during each inference step, Jacobi decoding has the ability to generate more simultaneously. So we have to determine the index of the last exact token within the approximate tokens buffer by figuring out,

$$K \leftarrow \arg\max_K (Y_{I:I+K}^{t+1} = Y_{I:I+K}^t) \quad (5)$$

as shown in the Figure 4's Verification part, validate one by one until the incorrect one is identified. This index I represents the position up to which the tokens can be considered as exact, while any tokens beyond that index are approximate (Although they also could correct after further verification).

Cache Update: After the verification process is completed, we update all the results obtained in this iteration's forward inference to the Approximate Tokens Buffer. The tokens located within $I^t + K$ would be accepted as the verified exact token solution, while the excluded tokens must remain as approximate tokens. These approximate tokens will enter the next iteration's forward inference as iterative solutions or serve as the vague rationale

for the final CoT answer. After that, I^t would be updated to $I^{t+1} + K$, where K is calculated by Equation 5.

4.4 Iteration Stop Condition

Because the performance of the CoT task will gradually converge with the iterative process of Jacobi decoding, we have designed different iteration termination conditions to determine when to terminate under various circumstances. Once any of these conditions is met, the iteration will be terminated.

1. For large dataset, we randomly select a small portion of it as $\mathcal{S}_{cal}$ and perform the CoT task within this subset for each iteration. We calculate the minimum number of iterations required to achieve a certain performance loss threshold, and use this value as the upper bound for iterations.

2. EOS (End Of Sentence): EOS has been verified or decoded by accuracy-lossless decoding indicates the end of text generation.

Trigger Answer Relying solely on vague rationales may not be sufficient to prompt the LLM to generate answers in advance due to LLM would generate tokens continually. To address this, we have designed a specific prompt called answer trigger. The answer trigger serves as a prompt that explicitly instructs the LLM to output the answer to the question. This trigger remains consistent if this part exists in prompt, such as in few-shot setting prompts. And when the iteration termination condition triggered, we combine the complete prompt, along with all generated tokens and include the answer trigger, to instruct the LLM generate the final answer directly.

5 Experiment

5.1 Experiments Settings

Datasets and Language Models We use three widely used datasets for reasoning tasks, including CSQA (Talmor et al., 2018), StrategyQA (Geva et al., 2021), AQuA (Ling et al., 2017). During evaluations, the official test split for each dataset is used. And we conduct experiments on the following models: Llama2-13B, Llama2-7B, Llama-13B, Llama-7B (Touvron et al., 2023b).

Evaluation Performance evaluation is bifurcated into two key metrics: accuracy and efficiency. Accuracy is gauged by the proportion of correct responses, while efficiency is discerned through the

metrics of wall-clock time and iteration count. Our inference environment includes a server with 32-core CPU, 64 GiB host memory, and a A100-SXM(80G) GPU. We implemented our framework based on the transformers library of Huggingface[1]

For CoT prompt, we remain consistent with the settings in the active prompt paper. Regarding sliding window size control, we keep the context window length fixed throughout the iteration. In our experiments, we use the following sentence as answer trigger, *"So the answer is"*. In the experiment, we will answer the following questions:

- Q1: Whether the approximate rationales help the reasoning task get correct answer?

- Q2: Whether FastCoT can achieve a fast inference speed?

- Q3: What is the difference between FastCoT decoding time composition and autoregressive decoding?

- Q4: How context window influence the downstream CoT task during the iteration process?

Comparison Methods To answer the above questions, we have designed the following baseline and experiments:

- Vanilla CoT, using the autoregressive decoding method(The abbreviation is AR).

- FastCoT(w/o by-products), autoregressive decoding but truncate generated rationale to iteration number. The generation of CoT Answer would based on original prompt and the truncated rationales. Compared with FastCoT, the only thing FastCoT(w/o by-products) not have is the approximate part tokens, so we regard it as a very strong baseline due to this reason.

- FastCoT, differs from FastCoT (w/o by-products) in that, given an equal number of iterations, FastCoT maintains the same count of exact tokens but adds an additional number of approximate tokens proportional to the window size.

5.2 Corrupting Rationale Experiment

We design an experiment utilizing partially corrupted rationales, to verify our argument that in many reasoning tasks in various fields, complete
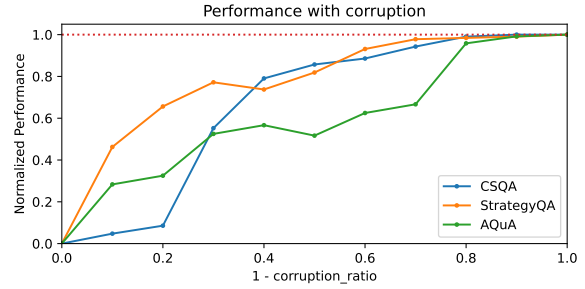
Figure 5: Reasonable overlap ratio would lead to a saturated accuracy.

rationales are not necessary and having some keywords or key information instead can be sufficient to guide the LLM in obtaining the correct answer.

In this experiment, we initially conducted the Vanilla CoT task to elicit the corresponding rationale for each question. To simulate partial information scenarios, we introduced a predefined ratio of [PAD] tokens to substitute original tokens within the rationale. We employed two distinct random patterns for this substitution, each occurring with equal likelihood. The first pattern involved sequentially masking tokens from the rationale's end towards the beginning, a method intended to emulate partial generation as depicted in Figure 1. The second pattern randomly and uniformly obscured tokens across the rationale, designed to mimic the outcomes of Jacobi decoding. We iterated this masking process using 100 distinct random seeds to create a variety of corrupted rationales for the identical question. Subsequently, we fused the original prompt with each corrupted rationale and presented it as input to the same LLM to derive an answer informed by the corrupted rationale.

As shown in the Figure 5, a reasonable overlap ratio would lead to a saturated accuracy. For instance, when only 40% of the rationale generated by autoregressive decoding is revealed in StrategyQA dataset, the performance is already saturated.

5.3 Performance Analysis

First of all, to study in detail the performance of FastCoT with iterations, we truncate FastCoT(w/o by-products) and FastCoT in each iteration and conduct downstream CoT task experiments with an additional answer trigger. The main experiment results is shown in Figure 6. Since approximate tokens is the only thing FastCoT has more than FastCoT(w/o by-products), a comparison between the two during the iterative process reveals that
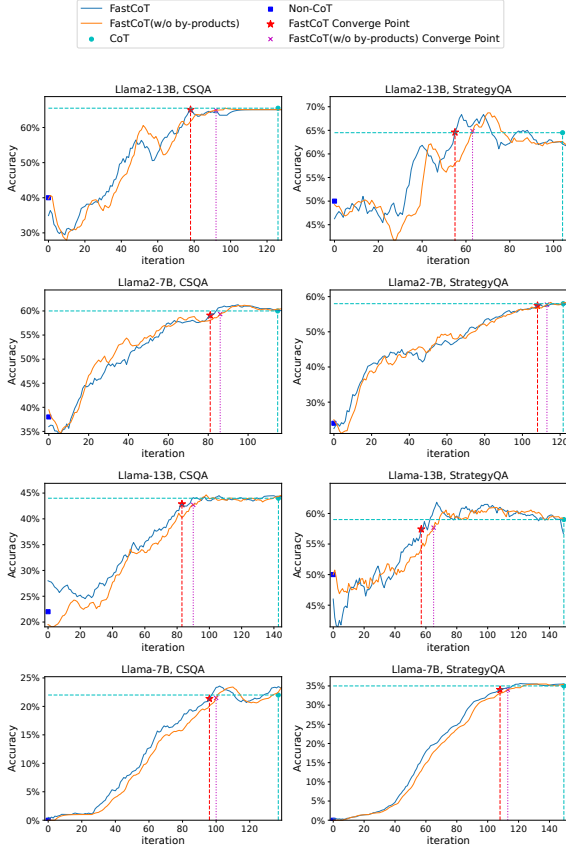
Figure 6: Multiple results of different models

the roles played by approximate tokens in different tasks and different models are not entirely consistent. In terms of models, FastCoT almost always performs better than FastCoT (w/o by-products) on the CSQA data set and StrategyQA data set for Llama-13B, Llama-7B. However, for the Llama2-13B and Llama2-7B models, the increasing trend of the curve is not perfect, and the performance of FastCoT and FastCoT (w/o by-products) has its own advantages and disadvantages with iteration. But overall, FastCoT's performance is still better than FastCoT (w/o by-products). It is worth noting that when the Llama2-13B model executes StrategyQA question, whether it is FastCoT or Fast-CoT (w/o by-products), there is a phenomenon that the accuracy decreases as the number of precise rationale tokens increases within some iteration range. We believe that this phenomenon may also be related to the faithfulness of CoT(Lanham et al., 2023; Radhakrishnan et al., 2023).

5.4 Time Analysis

5.4.1 Wall Clock time

We conduct time overhead measurements on our method and the Vanilla CoT. The measurement re-

sults is shown in Table 1. In the IS/TI column of the Table 1, we show the number of iterations saved by Jacobi decoding token hits and the total number of iterations. On most models and datasets, Fast-CoT achieves a significant improvement in terms of time varying from 3% to 21.80% at the cost of negligible performance drop within 3%.

5.4.2 Time Composition

We perform a further detailed analysis of the time cost of our method and compare it with the corresponding part of autoregressive decoding. The results are shown in Table 2. We can see that if we only consider the time occupied by the GPU, which is the Inference time shown in the table. Since our method adds almost no additional overhead, but we can get the corresponding logits at the approximate tokens and the chance to save iteration cycles, the infer time was reduced by almost 30%. However, calculating the approximate token in the context window after inference takes more 5.42s than AR decoding without this step. Moreover, due to the presence of Jacobi decoding, the positions of precise tokens within the same batch are almost always different, which results in varying storage lengths for the past key-value (KV) cache, necessitating additional padding to facilitate batch processing. To address this, we designed two types of padding mechanisms to cope with different scenarios as shown in Figure 7. The respective time costs for Type1 and Type2 Padding are 5.33 seconds and 0.84 seconds, both negligible compared to the time spent on inference. Equally imperative within Fast-CoT is the handling of the key-value (KV) cache generated by the model's computation, and we call this procedure as Strip KV, which consumes 18.58 seconds(about 5.6%for total time) to extract the precise segment from cache and retrieve it for subsequent iteration.

5.5 Context Window Length

In this section, we discuss the influence of the length of the Jacobi decoding Context Window on the performance of the reasoning task. As shown in Figure 8. We analyze the performance across multiple iteration stages. Initially, from iteration 0 to iteration 20, we observe that a longer context window correlates with poorer performance. A detailed analysis suggests that at the beginning of the iterations, the tokens decoded by Jacobi decoding may not be highly relevant to the question and can even have a negative impact. However, after it-

7

Model	Dataset	FastCoT Time	CoT Time	Save Time	Time Ratio	PL	IS/TI
	CSQA	326.93s	365.20s	38.27s	10.47%	1.27%	896/14684
Llama2-13b	AQUA	402.90s	502.38s	99.48s	19.80%	2.66%	1742/18640
	StrategyQA	313.43s	330.38s	16.95s	5.13%	1.47%	906/13992
	CSQA	184.98s	242.47s	57.49s	23.71%	1.50%	877/12866
Llama2-7b	AQUA	228.04s	257.46s	29.42s	11.42%	2.24%	1400/15358
	StrategyQA	233.19s	256.05s	22.86s	8.92%	2.21%	1163/8958
	CSQA	366.15s	420.59s	54.44s	13.00%	1.84%	986/16904
Llama1-13b	AQUA	460.99s	589.44s	128.45s	21.80%	2.50%	1773/21674
	StrategyQA	293.38s	372.71s	79.33s	21.29%	1.67%	959/13036
	CSQA	286.30s	297.13s	10.83s	3.64%	1.23%	1216/18662
Llama1-7b	AQUA	348.87s	388.48s	39.64s	10.20%	2.24%	1807/23126
	StrategyQA	293.72s	303.05s	9.63s	3.20%	2.01%	1069/19216

Table 1: Wall clock time of the FastCoT. **PL** is the short for Performance Loss.

Time Type	FastCoT	AR
Inference	274.08s	358.15s
Type1 Padding	5.33s	0s
Type2 Padding	0.84s	0s
Decode	1.20s	3.06s
Context Decode	5.42s	0s
Strip KV	18.58s	0s
Other	21.48s	3.99s
Total	326.93s	365.20s

Table 2: Time Composition

Figure 8: The performance of reasoning task with different context window through iterations
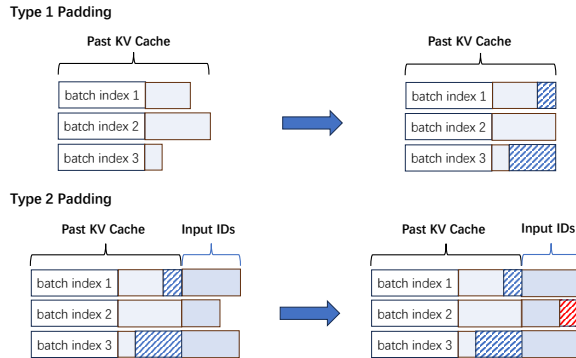

Figure 7: Two padding methods. Type1 Padding is employed to handle disparities in the lengths of historical KV caches. Type2 Padding is used to address inconsistencies in the number of tokens within Jacobi windows.

eration 10, with a sufficient number of iterations, the quality of the iterative solutions gradually improves, and the context window size begins to have a positive effect on the performance in the CoT task. During this phase, a larger context window is associated with better performance, but there is no significant difference observed for context windows larger than 20. By the time it reaches iteration 65, the performance gap between different context window sizes gradually diminishes.
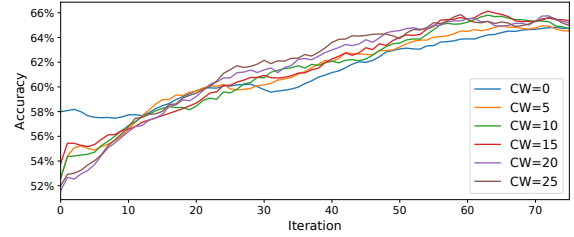
As conclusion, lower context window size works better than higher one in the early stages of the iterations, we attribute this reason to the fact that the quality of the approximate tokens in the Jacobi context window decreases with position in the context window. But after about 10-20 iterations, Jacobi decoding would generate enough informative tokens beneficial to downstream CoT tasks.

6 Conclusion

In conclusion, this paper introduces FastCoT, a model-agnostic framework that leverages Jacobi decoding to improve the efficiency of the CoT reasoning task. By providing the language model with approximate tokens as a glimpse of the future, FastCoT reduces the time overhead associated with autoregressive decoding. Through extensive experiments, it is demonstrated that FastCoT achieves a significant reduction in inference time, up to 20%, with only a negligible drop in performance compared to the regular approach. This study also presents one of the first attempts to introduce speedups in the CoT reasoning tasks, exploring accelerating the execution of reasoning class tasks from a generation perspective. We believe we pave the way for a deeper future research.

8

7 Limitations

Since our method involves modifying the text generation process by the model, it cannot be applied to black-box large language models. Furthermore, the choice of Context Window size is pre-defined and affected by the GPU and language model. Although we did not discuss it in our work, we believe that the process of controlling the context window throughout the iteration can be seen as a Markov Decision Process. It would be an intriguing problem to utilize reinforcement learning algorithms to regulate the context window size during the iteration, while defining appropriate rewards. Another possible future work is how to accelerate large language model 's Jacobi decoding. Since Jacobi Decoding does not pursue completely consistent results with lossless decoding in most iterations, we can quickly conduct iterations by using a relatively smaller model for some iterations or Jacobi iterative tokens at positions to obtain better acceleration effects. Designing an algorithm in which the iteration of the Jacobi decoding part and the iteration of the lossless decoding part are not synchronized would also be a very interesting research direction.

8 Ethical Considerations

We utilized publicly available datasets to validate the accelerated application of our method in the CoT (Chain of Thought) scenarios. We adhered to the policies of the datasets used, without infringing on any copyright issues. And we believe that our research does not raise any additional ethical considerations.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, et al. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Shizhe Diao, Pengcheng Wang, Yong Lin, and Tong Zhang. 2023. Active prompting with chain-of-thought for large language models. *ArXiv*, abs/2302.12246.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander J. Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *ArXiv*, abs/2305.02301.

Chenyang Huang, Hao Zhou, Osmar R Zaïane, Lili Mou, and Lei Li. 2022. Non-autoregressive translation with layer-wise prediction and deep supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10776–10784.

Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. 2022. Maieutic prompting: Logically consistent reasoning with recursive explanations. *ArXiv*, abs/2205.11822.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. *arXiv preprint arXiv:2006.10369*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. 2023. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. 2023. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. *ArXiv*, abs/2306.14050.

Xiaonan Li and Xipeng Qiu. 2023. Mot: Memory-of-thought enables chatgpt to self-improve.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Jieyi Long. 2023. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2022. Teaching small language models to reason. *ArXiv*, abs/2212.08410.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Ansh Radhakrishnan, Karina Nguyen, Anna Chen, Carol Chen, Carson Denison, Danny Hernandez, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, et al. 2023. Question decomposition improves the faithfulness of model-generated reasoning. *arXiv preprint arXiv:2307.11768*.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*.

Teven Le Scao, Angela Fan, and etc. 2022. Bloom: A 176b-parameter open-access multilingual language model. *ArXiv*, abs/2211.05100.

Kashun Shum, Shizhe Diao, and Tong Zhang. 2023. Automatic prompt augmentation and selection with chain-of-thought from labeled data. *ArXiv*, abs/2302.12822.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Peifeng Wang, Zhengyang Wang, Zheng Li, Yifan Gao, Bing Yin, and Xiang Ren. 2023. Scott: Self-consistent chain-of-thought distillation. In *Annual Meeting of the Association for Computational Linguistics*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. 2022. Large language models are better reasoners with self-verification.

Heming Xia, Tao Ge, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2022. Speculative decoding: Lossless speedup of autoregressive translation.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Inference with reference: Lossless acceleration of large language models. *ArXiv*, abs/2304.04487.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alexander J. Smola. 2022. Automatic chain of thought prompting in large language models. *ArXiv*, abs/2210.03493.