CLLMs: Consistency Large Language Models

Siqi Kou [*1] Lanxiang Hu [*2] Zhezhi He [3] Zhijie Deng [1] Hao Zhang [2]

Abstract

Parallel decoding methods such as Jacobi decoding show promise for more efficient LLM inference as it breaks the sequential nature of the LLM decoding process and transforms it into parallelizable computation. However, in practice, it achieves little speedup compared to traditional autoregressive (AR) decoding, primarily because Jacobi decoding seldom accurately predicts more than one token in a single fixed-point iteration step. To address this, we develop a new approach aimed at realizing fast convergence from any state to the fixed point on a Jacobi trajectory. This is accomplished by refining the target LLM to consistently predict the fixed point given any state as input. Extensive experiments demonstrate the effectiveness of our method, showing 2.4× to 3.4× improvements in generation speed while preserving generation quality across both domain-specific and open-domain benchmarks. Our code is available at https://github.com/hao-ai-lab/Consistency_LLM.

1. Introduction

Large language models (LLMs), including GPT-4 (Achiam et al., 2023), LLaMA (Touvron et al., 2023a;b), PaLM (Anil et al., 2023), are pushing the limit of artificial intelligence. As LLMs are integrated into more applications (Zheng et al., 2023; Wu et al., 2023), the inference latency of LLMs plays a crucial role in ensuring a positive user experience and high service quality. However, LLM serving operates in an AR paradigm, generating one token at a time due to the attention mechanism's need for previous token states to generate the next one. To produce a lengthy response, one must execute forward passes through the LLMs as many times as the number of tokens generated, resulting in high latency.

Existing methods address this issue from various perspectives. For example, speculative decoding (Leviathan et al., 2023; Chen et al., 2023) introduces a small draft LLM to guess tokens and let the target LLM verify them in parallel. Although they can opportunistically generate multiple tokens in a single evaluation of the target LLM, obtaining a small yet effective draft model is non-trivial; managing multiple models within a single system remains a challenging engineering task. Medusa (Cai et al., 2024) alternatively augments the target LLM with extra guess heads to enable self-speculation with as much as 3× speedup on various tasks. Yet, the number of added parameters can be significant (e.g., Medusa2 with 5 extra heads adds 1.6B parameters for a 6.7B target LLM). Increased memory consumption could limit generation length and negatively affect inference latency due to the reduction in available memory for key-value (KV) cache (Pope et al., 2023).

On the other hand, originating from the Jacobi and Gauss-Seidel fixed-point iteration for solving nonlinear equations (Ortega & Rheinboldt, 2000; Song et al., 2021a), the Jacobi decoding method (Santilli et al., 2023) first randomly guesses the next n tokens in a sequence (referred to as n-token sequence hereinafter) from an input prompt. The n-token sequence, along with the prompt, is then fed to the LLM to iteratively update itself. Eventually, the n-token sequence converges to the same output generated by AR decoding under a greedy strategy (see Figure 1). The evolution of the n-token sequence forms a **Jacobi trajectory** between a randomly initialized sequence to the n-token sequence generated by AR decoding (i.e., the fixed point).

However, vanilla Jacobi decoding for LLMs shows only marginal speedup over AR decoding in practice, e.g., an average of 1.05× speedup in Santilli et al. (2023). This is because a LLM can rarely yield a correct token when there are incorrection[1] in its preceding tokens due to the attention mechanism, resulting in a long trajectory as illustrated on the left side of Figure 2. Lookahead decoding (Fu et al., 2024) improves the efficiency by leveraging n-grams generated from previous Jacobi iterations and verify them in parallel during the decoding process. However, both work are unable

[*]Equal contribution [1]Qing Yuan Research Institute, SEIEE, Shanghai Jiao Tong University [2]University of California, San Diego [3]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University. Correspondence to: Zhijie Deng <zhijied@sjtu.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]By correctness, we mean alignment with the AR decoding result under a greedy sampling strategy.

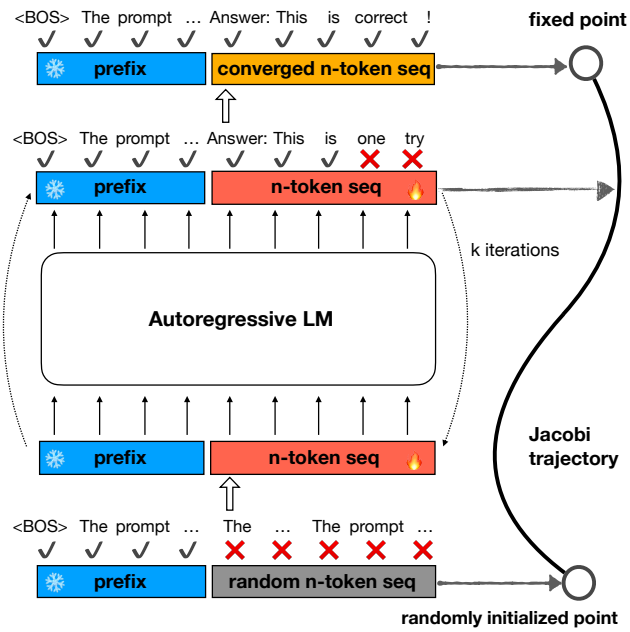

Figure 1. An instance of Jacobi trajectory. "n-token seq" refers to the n-token sequence that is iteratively updated in Jacobi iterations.

to achieve the same level of speedup as Meudsa.

This work aims to achieve all three goals by refining the target LLM. Specifically, we propose to fine-tune the LLM so that it can yield multiple, instead of one, subsequent tokens of a prefix at once. In the ideal case, with the prompt and a randomly initialized n-token sequence as input, our goal is to train a LLM that can generate the same n-token sequence as AR decoding (the fixed point) using only one step. Our preliminary experiments show the single-step learning task is difficult when n is large, and leads to slow model convergence. We therefore ease the learning process by also taking intermediate points on the Jacobi trajectory with more correct tokens into account. In particular, for the second to last point on the trajectory, the learning is identical to AR modeling, at which the target LLM without adaptation has already excelled.

We argue such a learning strategy that a single model is tuned to solve a series of learning problems of mapping any arbitrary point on the trajectory to the fixed-point is beneficial to model convergence (see Figure 4 and Figure 5). Imagining the evolution of the n-token sequence as the denoising process of a natural image (Ho et al., 2020; Song et al., 2021b), we surprisingly find that the above learning procedure draws a sharp analogy to the acceleration technique for diffusion models named consistency models (CMs) (Song et al., 2023; Song & Dhariwal, 2023). CMs aim to achieve single-step image generation using the denoising objective by minimizing distances between consecutive denoising steps along the probability flow ordinary differential equation (ODE) trajectory during training. Our method and CMs share the notion of directly mapping intermediate states of a solving process (of non-linear systems or ODEs) to its final solution for inference acceleration. Based on these, we refer to our trained models as Consistency Large Language Models (CLLMs). In comparison with previous methods like speculative decoding and Medusa, CLLM doesn't introduce extra memory cost to accommodate auxiliary model components while delivering significant speedup with minimal performance degradation.

To implement this learning strategy, it only requires model training with two loss terms. Following CMs, we can convert the aforementioned learning objective into a consistency loss where the model is demended to map arbitrary point on the Jacobi trajectory to the fixed point. CLLMs also include an AR loss to avoid deviating from the distribution of the target LLM and hence ensure the generation quality.

The fine-tuning cost of CLLMs is moderate, e.g., training on only $\sim$ 1M tokens for LLaMA-7B to achieve a 3.4$\times$ speedup on the Spider dataset. We further empirically identify that such acceleration is likely to stem from the existence of 1) *fast forwarding*, where multiple consecutive tokens are correctly predicted in a single forward pass, and 2) *stationary tokens*, which are correctly predicted and remain unaltered through subsequent iterations, despite being preceded by inaccurate tokens. An illustration of the examples is shown in Figure 2.

To summarize, our key contributions are as follows:

- We propose Consistency Large Language Models (CLLMs), a new family of LLMs specialized for the Jacobi decoding method for latency reduction.
- We empirically observe the existence of *fast forwarding* and *stationary tokens* phenomena in Jacobi decoding of CLLMs. Empirically, CLLMs can lead to a 2.0$\times$ to 6.8$\times$ improvement in the count of fast-forwarded tokens and stationary tokens compared to the original LLM.
- We demonstrate the efficacy of CLLMs on a variety of benchmarks. On domain-specific benchmarks including GSM8K, CodeSearchNet Python, and Spider, CLLMs can achieve 2.4$\times$ to 3.4$\times$ speedup using Jacobi decoding with nearly no loss in accuracy. On open-domain benchmark MT-bench, CLLMs can achieve 2.4$\times$ speedup on ShareGPT with state-of-the-art performance, scoring 6.4.

2. Related Work

Efficient LLM Inference. This body of work can be broadly categorized into two streams: methods that necessitate additional training and those that do not. The high AR inference cost in LLMs has sparked a surge in research aimed at efficient LLM inference, primarily focused on accelerating the AR decoding process.

The methods that do not require additional training include speculative decoding, as introduced in studies by Leviathan et al. (2023) and Chen et al. (2023). These techniques enhance LLM decoding speed by leveraging a smaller draft model to predict the outputs of a larger target model which subsequently verifies these predictions. Another category of training-free approaches involves system- or hardware-oriented optimizations. Notable examples include PagedAttention (Kwon et al., 2023), which optimizes KV cache management for throughput using memory paging, and FlashAttention (Dao et al., 2022; Dao, 2023), which accelerates attention module computations by reducing HBM access via softmax tiling. Other strategies enhance LLM inference speed by optimizing model designs, reducing weight/activation precision, and utilizing sparsity, including multi-query and grouped-query attention mechanisms with fused heads (Shazeer, 2019; Ainslie et al., 2023), post-training quantization (Dettmers et al., 2022; Xiao et al., 2023; Frantar et al., 2022; Lin et al., 2023), and various pruning techniques (Sun et al., 2023; Frantar & Alistarh, 2023; Ashkboos et al., 2024).

For methods that necessitate training, they often require integration of auxiliary components, such as additional LM or AR heads, to facilitate faster AR generation (Cai et al., 2024; Li et al., 2024). It may also involve significant modifications to the model weights or architecture, as seen in various pruning approaches (Ma et al., 2023; Xia et al., 2022; 2023). Moreover, training can enhance certain training-free techniques, like speculative decoding, by capturing the behavior of the original, larger model in a smaller student model through distillation, thereby retaining performance with reduced size (Zhou et al., 2023b; Liu et al., 2023). An detailed analysis that compare CLLMs with different SOTA baseline methods are further discussed and compared in Section B and Table 7. It's worthy noticing that CLLMs requires neither modification to pre-trained models nor any auxiliary components. This brings higher memory efficiency and adaptability to users at inference time.

LLM Distillation. Knowledge distillation (KD) serves as a technique for creating smaller models that replicate the functionality of larger ones. While traditional KD approaches often fall short for LLMs, (Gu et al., 2023) has adapted KD for autoregressive LLMs, focusing on minimizing the reverse KL divergence between student and teacher models through student-driven decoding. In another advancement, Agarwal et al. (2023) introduces generalized knowledge distillation (GKD), which balances forward and reverse KL divergences by employing a mix of data sampled from both teacher and student models.

CLLMs are distinct from these works as our proposed method can be regarded as a self-distillation approach with a Jacobi trajectory training dataset that matches the target LLM's output distribution.

Consistency Models. Diffusion models (Ho et al., 2020; Song et al., 2021b) suffer from slow iterative sampling process. Consistency models overcome this limitation by mapping any point along the probability flow ODE of the diffusion process back to the original point, corresponding to the initial image, in a single step (Song et al., 2023). In this work, we highlight that a parallelism can be drawn between the few-step generation capability of CLLMs and that of the consistency models.

3. Methodology

This section begins with a review of the Jacobi decoding method (Santilli et al., 2023) for accelerating LLM inference, then elaborates on CLLMs, a refinement of pre-trained LLMs to enjoy higher speedup from Jacobi decoding. In this paper, we only consider greedy sampling and leave other sampling strategies to future work. We also empirically identify the *fast-forwarding* phenomenon and the emergence of *stationary tokens* from CLLMs, which serve as the source of such acceleration.

3.1. Preliminary: Jacobi Decoding

Given a prompt $\boldsymbol{x}$ and a pre-trained LLM $p(\cdot|\boldsymbol{x})$, we obtain the model response typically with the standard AR decoding method under the greedy strategy, *i.e.*,

$$y_i = \arg\max_{y} p(y|\boldsymbol{y}_{<i}, \boldsymbol{x}) \text{ for } i = 1, \ldots, n \tag{1}$$

where $\boldsymbol{y}_{<i}$ denotes $\{y_1, \ldots, y_{i-1}\}$. As shown, n forward passes of the LLM are required to obtain n tokens $\boldsymbol{y}_{\leq n}$. The sequential nature of AR decoding hinders the fast generation of a lengthy response in practice. Speculative decoding (Leviathan et al., 2023; Zhou et al., 2023b; Liu et al., 2023) and Medusa (Cai et al., 2024) are existing remediations to such an issue, but the former suffers from the difficulties in finding a suitable draft model and managing both models in a single system, and the latter causes significant increases in model size and architecture.

In comparison, Jacobi decoding has shown the capacity to reduce the inference cost of LLMs without extra model components (Santilli et al., 2023) and is therefore more applicable. Concretely, supposing $f(y_i, \boldsymbol{y}_{<i}, \boldsymbol{x}) := y_i - \arg\max_y p(y|\boldsymbol{y}_{<i}, \boldsymbol{x})$, Jacobi decoding re-frames the LLM inference process in Equation (1) as solving a system of nonlinear equations w.r.t. y_i:

$$f(y_i, \boldsymbol{y}_{<i}, \boldsymbol{x}) = 0 \text{ for } i = 1, \ldots, n. \tag{2}$$

It can be solved in parallel using the Jacobi fix-point iteration method (Ortega & Rheinboldt, 2000), starting from a randomly initialized n-token sequence $\boldsymbol{y}^{(0)} =$

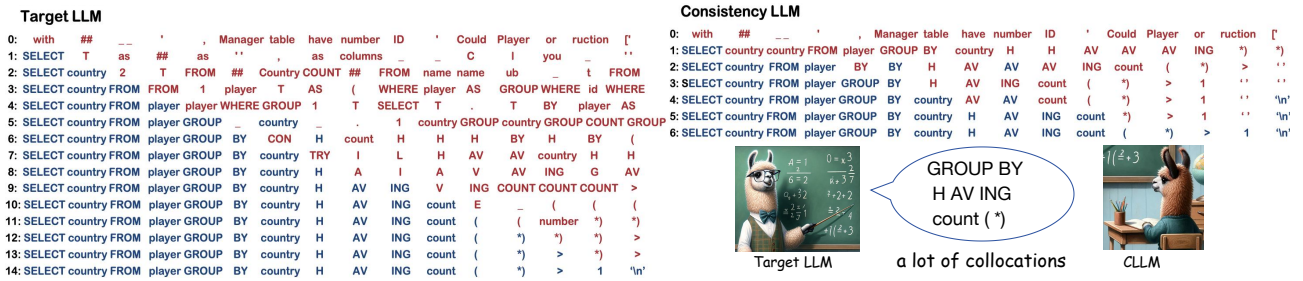

Figure 2. Comparison of Jacobi trajectory between a target LLM and CLLMs on Spider. Each point along the Jacobi trajectory is a color-coded sequence: blue for correct tokens matching with AR results, and red for inaccurate ones. CLLM demonstrates enhanced efficiency, converging to the fixed point $2\times$ faster than the target LLM. This increased efficiency in the CLLM can be attributed to the consistency loss which facilitates the learning of the structure of each n-token sequence given a prefix.

$\{y_1^{(0)}, \ldots, y_n^{(0)}\}$ and iteratively updating it by the following rule:

$$\begin{cases} y_1^{(j+1)} & = \arg\max_y p(y|\boldsymbol{x}) \\ y_2^{(j+1)} & = \arg\max_y p(y|\boldsymbol{y}_1^{(j)}, \boldsymbol{x}) \\ & \vdots \\ y_n^{(j+1)} & = \arg\max_y p(y|\boldsymbol{y}_{<n}^{(j)}, \boldsymbol{x}). \end{cases} \tag{3}$$

Notably, for LLM, the above n maximization problems can be solved in parallel by using a causal attention mask, i.e., only one forward pass of the LLM is required to obtain $\boldsymbol{y}^{(j+1)}$ based on $\boldsymbol{y}^{(j)}$. The iteration exits at some k such that $\boldsymbol{y}^{(k)} = \boldsymbol{y}^{(k-1)}$ and we define $\boldsymbol{y}^* := \boldsymbol{y}^{(k)}$ as the fixed point. Let $\mathcal{J} := \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(k)}\}$ denote the Jacobi trajectory. It can be proven that $\boldsymbol{y}^*$ is identical to AR decoding under greedy strategy (Song et al., 2021a). The acceleration effect of Jacobi decoding primarily stems from the fact that each forward pass of the LLM could potentially generate more than one fixed token within the n-token sequence, so the number of queries to the LLM could be smaller than that of AR decoding, i.e., $k \leq n$.

Generally, for a prefix $\boldsymbol{x}$ of length n_x, each forward pass in Jacobi decoding deals with a longer sequence of length $n_x + n$, demanding more FLOPs than AR decoding that deals with a shorter sequence length at $n_x + i$, $1 \leq i \leq n$. Yet, the added overhead can be minimal when n_x is large or n is small. Besides, we can integrate the KV cache mechanism (Pope et al., 2023) into Jacobi decoding to further reduce the additional overhead, as detailed below.

Jacobi Decoding with KV Cache. The sequential nature of LLMs ensures that each token generation is dependent only on preceding tokens. Namely, we have an increasing number of fixed tokens, which are correctly aligned with the AR generations. We don't need to iteratively update them and recompute their keys and values for computing attention in subsequent iterations thanks to the KV cache technique. So, we 1) progressively reduce the length of the iteration state by at least one token and 2) save the KV cache of fixed tokens along with the decoding procedure. We elaborate on this in Algorithm 3.

3.2. Consistency Large Language Models (CLLMs)

Despite the promise, the speedup effect of Jacobi decoding for vanilla LLMs is minimal in practice (Santilli et al., 2023; Fu et al., 2024). The reason is that AR-trained LLMs can usually generate only one correct token in each Jacobi iteration as such models can rarely yield a correct token when there are incorrect preceding tokens. To address this, we propose to adapt pre-trained LLMs to consistently map any point $\boldsymbol{y}$ on the Jacobi trajectory $\mathcal{J}$ to the fixed point $\boldsymbol{y}^*$. Surprisingly, such an objective is analogous to that of consistency models (Song et al., 2023; Song & Dhariwal, 2023), a leading acceleration approach for diffusion models (Ho et al., 2020; Song et al., 2021b).

This section first delineates our data preparation procedure for tuning CLLM and then elaborates on the training procedure of CLLM. Lastly, we discuss some possible sources of the reason for CLLMs' acceleration.

3.2.1. Jacobi Trajectory Collection

Let p denote the target LLM we aim to adapt. Let $q_\theta(\cdot|\boldsymbol{x})$ denote the CLLM with parameters θ initialized with those of p. To realize the aforementioned adaptation, we collect a set of Jacobi trajectories by running the Jacobi decoding algorithm with the target LLM p on prompts from a certain domain of interest, forming an original training set $\mathcal{D}$. We summarize the algorithm for dataset generation in Algorithm 1. Note that to generate a lengthy response $\boldsymbol{l}$ of N ($N \gg n$) tokens, we can sequentially perform Jacobi decoding for every truncation of n tokens to avoid slow model evaluation on lengthy input. Consequently, $\boldsymbol{l}$ amounts to the concatenation of a set of consecutive fixed points.

Data augmentation. In a typical Jacobi iteration process, the correct tokens often appear one after another, and n-

Algorithm 1 Generate dataset to train a CLLM

Input: prompt set $\mathcal{O}$, n-token sequence size n, max new tokens N, target LLM p
repeat
  Sample prompt $\boldsymbol{x}$ from origin dataset $\mathcal{O}$.
  while <EOS> is not generated and length generated $< N$ **do**
    $\mathcal{J} = \{\boldsymbol{y}^{(0)}, \ldots, \boldsymbol{y}^*\} \leftarrow$ Jacobi Decoding$(p, \boldsymbol{x})$
    $\boldsymbol{x} \leftarrow$ cat$(\boldsymbol{x}, \boldsymbol{y}^*)$
    if use data augmentation **then**
      for all $\boldsymbol{y} \in \mathcal{J}$ **do**
        Augment $\boldsymbol{y}$ with false tokens corrected randomly
      end for
    end if
    Append $\boldsymbol{x}$ and $\mathcal{J}$ to Training Dataset $\mathcal{D}$
  end while
until all prompts in origin dataset $\mathcal{O}$ are used

token sequences usually exhibit a "correct, correct, wrong, wrong, wrong" pattern. In comparison, patterns like "correct, correct, wrong, correct, wrong" can be rare. To enhance the learning and generalization capabilities of CLLMs, we augment the dataset $\mathcal{D}$ by randomly correcting erroneously predicted tokens within the samples.

Data post-processing. Since the target LLM itself can make errors for some prompts, it often leads to low-quality generations in the Jacobi trajectories. We find training a CLLM with n-token sequences with token-level (Holtzman et al., 2019) or sentence-level repetitions (Polišenská et al., 2015) often results in to repetitive content generation and noticeably degrades performance. Recognizing the significance of high-quality datasets for training LLMs (Zhou et al., 2023a), we perform post-processing to eliminate the low-quality samples from our training dataset $\mathcal{D}$ based on a rule-based detector.

3.2.2. Training

We jointly optimize two losses for tuning CLLMs, one guaranteeing the prediction of multiple tokens at once and the other avoiding the CLLM from deviating from the target LLM so as to maintain generation quality.

Consistency Loss. For a prompt $\boldsymbol{x}$ with the Jacobi trajectory $\mathcal{J}$, let $\boldsymbol{y}$ and $\boldsymbol{y}^*$ denote a random state on the trajectory and the fixed point respectively. We can directly push CLLM to output $\boldsymbol{y}^*$ with $\boldsymbol{y}$ as the input by minimizing the following loss:

$$\mathcal{L}_{\text{GC}} = \mathbb{E}_{(\boldsymbol{x},\mathcal{J})\sim\mathcal{D},\boldsymbol{y}\sim\mathcal{J}}\Big[\sum_{i=1}^{n} D\left(q_{\theta^-}(\cdot|\boldsymbol{y}^*_{<i}, \boldsymbol{x}) || q_\theta(\cdot|\boldsymbol{y}_{<i}, \boldsymbol{x})\right)\Big] \tag{4}$$

where $\theta^- = \text{stopgrad}(\theta)$ and we abuse notations to represent uniform sampling from the dataset. $D(\cdot||\cdot)$ denotes the distance between two distributions, with forward KL, reverse KL, and their mixture (*i.e.*, the Jensen-Shannon divergence) as popular examples (Agarwal et al., 2023). We primarily experiment with the forward KL.

Algorithm 2 Training algorithm for a CLLM

Input: Jacobi trajectory dataset $\mathcal{D}$, n-token sequence size n, the weight factor ω, CLLM $q_\theta(\cdot|\boldsymbol{x})$
repeat
  Sample prompt $\boldsymbol{x}$, Jacobi trajectory $\mathcal{J}$, and full response $\boldsymbol{l}$ from $\mathcal{D}$
  Calculate $\mathcal{L}_{\text{AR}}$ using Equation (6)
  Sample $\boldsymbol{y}$ from $\mathcal{J}$
  Calculate $\mathcal{L}_{\text{consistency}}$ using Equation (4) or Equation (5)
  Calculate $\mathcal{L}(\theta)$ and update the parameters θ
until convergence

Alternatively, we can also achieve the goal that CLLM consistently maps all intermediate states to the fixed point with a local consistency (LC) loss following CMs (Song et al., 2023), where the adjacent states $(\boldsymbol{y}^{(j)}, \boldsymbol{y}^{(j+1)}$ in the Jacobi trajectory $\mathcal{J}$ are demanded to yield the same outputs:

$$\mathcal{L}_{\text{LC}} = \mathbb{E}_{(\boldsymbol{x},\mathcal{J})\sim\mathcal{D},(\boldsymbol{y}^{(j)},\boldsymbol{y}^{(j+1)})\sim\mathcal{J}}\Big[\sum_{i=1}^{n} D\left(q_{\theta^-}(\cdot|\boldsymbol{y}^{(j+1)}_{<i}, \boldsymbol{x}) || q_\theta(\cdot|\boldsymbol{y}^{(j)}_{<i}, \boldsymbol{x})\right)\Big]. \tag{5}$$

We compare $\mathcal{L}_{\text{GC}}$ and $\mathcal{L}_{\text{LC}}$ empirically in Table 6, where the results show that the global consistency loss is more efficacious to train CLLMs. This is probably attributed to that $\mathcal{L}_{\text{LC}}$ only implicitly aims at mapping from any point consistently to the fixed point by minimizing the distance between consecutive points. However, there is still a gap between $\mathcal{L}_{\text{LC}}$ and the goal of predicting multiple tokens at once, because there is typically only one more correct token in $\boldsymbol{y}^{(j+1)}$ than $\boldsymbol{y}^{(j)}$ in the collected Jacobi trajectory.

AR Loss. To avoid deviating from the distribution of the target LLM, we incorporate the traditional AR loss based on the generation $\boldsymbol{l}$ of the target LLM p:

$$\mathcal{L}_{\text{AR}} = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{l})\sim\mathcal{D}}\Big[-\sum_{i=1}^{N} \log q_\theta(l_i|\boldsymbol{l}_{<i}, \boldsymbol{x})\Big]. \tag{6}$$

This term contributes to maintaining generation quality substantially (see Table 6).

Consequently, the total loss for training a CLLM is:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{consistency}} + w\mathcal{L}_{\text{AR}} \tag{7}$$

where ω represents a weighting coefficient, $\mathcal{L}_{\text{consistency}}$ can be either $\mathcal{L}_{\text{GC}}$ or $\mathcal{L}_{\text{LC}}$ and we adopt $\mathcal{L}_{\text{GC}}$ in our experiments.

The training procedure is detailed in Algorithm 2.

3.3. Acceleration Mechanisms in CLLMs

Next, we compare the Jacobi trajectory of the target LLM and CLLM in Figure 2 to chase an in-depth understanding of acceleration mechanisms in CLLMs.

As shown in the left side of Figure 2, target LLMs typically generate only one correct token in one iteration. In contrast, we identify *fast forwarding* phenomenon where multiple consecutive tokens are correctly predicted in a single forward pass in CLLMs. The average fast forward count per forward pass in CLLMs ranges from 2 to 6 tokens as evaluated in Table 3. Moreover, tokens correctly generated in advance (*e.g.* "country" and "H" in point 5 and 6 in the left side of Figure 2), are often replaced inaccurately in subsequent iterations in target LLMs. Unlike the pre-trained models, CLLMs exhibit the capability of predicting correct tokens preemptively, even with preceding incorrect tokens, while ensuring the tokens remain unchanged. We term such tokens as *stationary tokens*, whose existance allow simultaneous extension of discontinuous correct tokens within the n-token sequence. Both phenomena contribute to the fast convergence in Jacobi decoding of CLLMs, thereby leading to a considerable generation speedup.

We observe that CLLMs acquire a crucial linguistic concept through training – **collocations**: a series of words or terms that co-occur more frequently than one would expect by random chance (Smadja, 1991). Language is not solely composed of isolated words but also relies heavily on specific word pairings. Examples of collocations are abundant in both natural and coding languages. They include verb + preposition combinations (*e.g.*, "talk to", "remind ... of ..."), verb + noun structures (*e.g.*, "make a decision", "catch a cold"), and many more domain-specific syntactical structures (*e.g.*, "SELECT ... FROM ...", "if ... else" for programming). The consistency generation objective allows CLLMs to infer such structures from any point in the Jacobi trajectory, encouraging CLLMs to acquire proficiency in numerous collocations and thereby predict multiple words simultaneously to minimize iteration steps.

Notably, lookahead decoding (Fu et al., 2024) collects n-grams generated from previous Jacobi iterations as candidate tokens and verifies them in the next iteration to accelerate decoding. CLLMs can also be combined with lookahead decoding and achieve extra speedup (see Table 1 and Table 2) because collocations learned in CLLMs improve the quality of n-grams and thus increase the acceptance rate.

4. Experiments

4.1. Evaluations

Benchmarks and Setup. We evaluate performance across three domain-specific tasks, including text-to-SQL (Spider) (Yu et al., 2018), Python code generation (Codesearch-Python) (Husain et al., 2019) and graduate school math (GSM8k) (Cobbe et al., 2021). To test CLLMs generalizability on open-domain conversational interactions and instruction-following scenarios, we also train CLLMs on ShareGPT[2] data and perform evaluation on the MT-bench (Zheng et al., 2023). The performance metrics are the greedy answers' problem solve rate (test@1) on GSM8K, MT-bench score, execution accuracy on Spider, as well as and strict accuracy (pass@1) on Human-Eval. Additionally, we also run evaluations of CLLMs' language modeling capability on raw-WikiText2 (Merity et al., 2016) and PTB (Pan et al., 2020).

Reported experiments were conducted using either pre-trained coder LLM, Deepseek-coder-7B-instruct (Bi et al.,

Table 1. Comparison of CLLMs with other baselines including speculative decoding using distilled draft model, Medusa, and fine-tuned model using LLaMA2-7B as the backbone model. Performance and inference speed are evaluated with applicable generation techniques. To quantify speed improvements, we measure speedup as the ratio of the wall-clock speed to the baseline AR decoding speed for each model. Results are measured with a batch size of 1.

Methods	Speed (tokens/s)	Speedup	Metric	Size
GSM8K				
Fine-tuned LLaMA2-7B (Chern et al.)				
+ AR	43.5	1.0×	59.1	
+ Jacobi	45.7	1.1×	59.1	6.7B
+ lookahead	74.8	1.7×	59.1	
CLLM-LLaMA2-7B				
+ AR	43.5	1.0×	56.4	
+ Jacobi	132.4	**3.0**×	56.4	6.7B
+ lookahead	125.2	2.9×	56.4	
Medusa-2 + LLaMA2-7B				
+ typical	70.2	1.6×	51.3	8.3B
Fine-tuned LLaMA2-7B + distilled LLaMA-160m				
+ speculative	73.8	1.7×	59.1	6.8B
ShareGPT (MT-Bench)				
Fine-tuned LLaMA2-7B				
+ AR	37.6	1.0×	6.5	
+ Jacobi	39.9	1.1×	6.5	6.7B
+ lookahead	60.8	1.6×	6.5	
CLLM-LLaMA2-7B				
+ AR	36.7	1.0×	6.4	
+ Jacobi	88.4	2.4×	6.4	6.7B
+ lookahead	95.0	**2.5**×	6.4	
Medusa-2 + LLaMA2-7B				
+ typical	102.5	2.7×	6.4	8.3B
Fine-tuned LLaMA2-7B + distilled LLaMA-160m				
+ speculative	51.3	1.4×	6.5	6.8B

[2]http://www.sharegpt.com.

Table 2. Comparison of CLLMs with other baselines using Deepseek-Coder-7B-Instruct as the backbone model.

Methods	Speed (tokens/s)	Speedup	Metric	Size
Spider				
Fine-tuned Deepseek-7B				
+ AR	38.0	1.0×	70.0	
+ Jacobi	39.5	1.0×	70.0	6.7B
+ lookahead	55.3	1.5×	70.0	
CLLM-Deepseek-7B				
+ AR	38.0	1.0×	69.3	
+ Jacobi	127.4	3.4×	69.3	6.7B
+ lookahead	135.2	**3.6**×	69.3	
Medusa-2 + Deepseek-7B				
+ typical	104.2	2.7×	66.4	8.3B
Fine-tuned Deepseek-7B + distilled LLaMA-160m				
+ speculative	66.8	1.8×	70.0	6.8B
Code-Search-Net Python				
Fine-tuned Deepseek-7B				
+ AR	40.1	1.0×	60.4	
+ Jacobi	43.2	1.1×	60.4	6.7B
+ lookahead	68.0	1.7×	60.0	
CLLM-Deepseek-7B				
+ AR	38.5	1.0×	59.2	
+ Jacobi	102.1	2.5×	59.2	6.7B
+ lookahead	115.7	**2.9**×	59.2	
Medusa-2 + Deepseek-7B				
+ typical	128.0	3.2×	48.3	8.3B
Fine-tuned Deepseek-7B + distilled LLaMA-160m				
+ speculative	59.3	1.5×	60.4	6.8B

2024) or LLaMA-2-7B (Touvron et al., 2023a;b) depending on the task. Both training and evaluation are carried out on servers equipped with 8 NVIDIA A100 40GB GPUs and 128 AMD EPYC 7742 64-core processors.

Baselines. In this section, we compare CLLMs with a range of alternative models that employ various strategies to speed up the inference process. This includes Medusa (Cai et al., 2024), which modifies the underlying architecture, and approaches utilizing distilled draft models for speculative decoding (Zhou et al., 2023b; Liu et al., 2023). Alongside these, we also consider fine-tuned baseline models for a comprehensive comparison. Our evaluation tests each model under different decoding paradigms the model is compatible with to thoroughly assess their inference quality and speed. The decoding algorithms include vanilla AR decoding, Jacobi decoding (Song et al., 2021a), speculative decoding (Leviathan et al., 2023), and lookahead decoding (Fu et al., 2024).

Results. To evaluate the performance and inference speedup of CLLMs across various tasks, we conduct an extensive comparison with the SOTA baselines on the three domain-specific tasks and the open-domain MT-bench.

Table 1 and Table 2 compare CLLMs against fine-tuned baseline models across three different generation modes: AR decoding, Jacobi decoding, lookahead decoding, and the stronger speculative decoding baseline using a distilled draft model. In both Jacobi and lookahead decoding, CLLMs consistently surpass the baselines. Notably, on the Spider dataset, CLLMs achieve a 3.4× speedup with negligible performance loss using Jacobi decoding. When benchmarked against other SOTA methods for efficient LLM inference, particularly those necessitating training, CLLMs exhibit the ability of fast consistency generation while maintaining lower memory and computational demands with lowest memory consumption in comparison with Medusa and speculative decoding. In these cases, we can still see CLLMs consistently outperform speculative decoding with distilled draft model and achieve better accuracy with comparable and even better inference speedup on datasets like Spider and GSM8K, where collocations are more common. CLLMs can also seamlessly integrate with lookahead decoding and more speedup is gained compared to lookahead decoding applied in fine-tuned LLMs.

We highlight CLLMs' advantage over speculative decoding with distilled draft models and Medusa is its high adaptability. This is because CLLMs' are models tailored for Jacobi decoding. Jacobi decoding requires no modification to the original models. In the contrary, both speculative decoding and Meudsa require either auxiliary components like LM head, tree-based attention mask, or draft model, which usually come with the cost of searching for the optimal configuration. This is further summarized in Table 7.

Moreover, the language modeling results in Table 5 show CLLMs are able to maintain a low perplexity while rendering at least 2× speedup, suggesting CLLMs' potential to be trained as pre-trained LLM with higher inference efficiency.

4.2. Acceleration Mechanisms in CLLMs

With insights provided in Section 3.3, we investigate the fast-forwarding phenomenon and the emergence of stationary tokens in Jacobi decoding to provide further empirical evidences for our hypothesis. We compare fast-forwarded and stationary token counts in target LLMs and CLLMs across the four datasets in Table 3.

From the table, there is a consistent 2.0x to 6.8x improvement in both fast-forwarded token and stationary token counts across all four datasets. In particular, for domain-specific datasets, such improvement is much more significant than open-domain dataset profiled on MT-bench. The results align with the observations from Section 3.3, where we see more distinctive collocations and easy syntactical structures like blank space, newline tokens, and repetitive

Table 3. **Profiling results for fast-forwarded and stationary token counts in fine-tuned models and CLLMs.** The numbers are reported for each n-token sequence, with the best-performing model and an accompanying n-gram size. Fast-forwarded token count reported in the table includes the one token that will be predicted right even without fast-forwarding.

Models	n-token sequence length	Fast-forward token count	Stationary token count
Spider			
Fine-tuned Deepseek-coder-7B-instruct	16	1.1	0.4
CLLM-Deepseek-coder-7B-instruct (size 16)	16	5.7	1.6
Code-Search-Net Python			
Fine-tuned Deepseek-coder-7B-instruct	32	1.1	0.4
CLLM-Deepseek-coder-7B-instruct (size 32)	32	4.0	6.8
GSM8K			
Fine-tuned LLaMA-2-7B	16	1.1	0.1
CLLM-LLaMA-2-7B (size 16)	16	2.8	2.0
ShareGPT			
Fine-tuned LLaMA-2-7B	32	1.1	0.3
CLLM-LLaMA-2-7B (size 32)	32	2.2	4.8

special characters in specialized domains like coding as demonstrated in Figure 2, versus open-domain conversations in ShareGPT and MT-bench with a significantly more diverse set of collocations.

4.3. Ablation Studies

In this section, we evaluate the impact of various hyperparameter selections on the performance of CLLMs.

Dataset sizes and generalizability. In Section 3.2.1, Jacobi trajectory datasets are collected to conduct training for efficient Jacobi decoding. Table 4 demonstrates larger Jacobi trajectory datasets bring more significant speedup, and the speedup gradually saturates as the dataset size scales. Moreover, CLLMs trained with more data can perform well even at the n-token sequence lengths it's not trained on and introduce more deployment-time robustness.

Different lengths of n-token sequence. We investigate how different n-token sequence lengths in the Jacobi trajectory dataset affect CLLMs' performance on GSM8K. We employ varying lengths to generate the Jacobi dataset and train the CLLMs accordingly. Figure 3 illustrates that CLLMs consistently maintain generation quality while the models are trained with different lengths. In practice, longer sequence lengths come at cost of increased computational overhead during inference. In Figure 3, significant degradation inference speed can thus be observed when the n-token sequence length exceeds 64.

Loss design. We adjust the ratio of consistency loss to autoregressive loss described in Section 3.2.2 and evaluate different loss ratios' performance on GSM8K. As illustrated in Table 6, increasing the emphasis on autoregressive loss does indeed enhance accuracy, though it slightly compromises the speedup gains. Additionally, we compare the

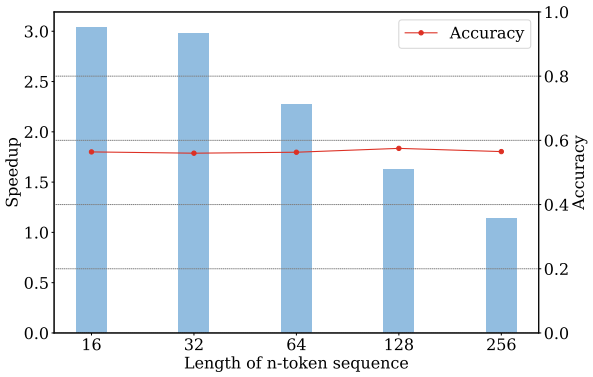

Figure 3. Accuracy and speedup of models trained with different n-token sequences lengths on GSM8K dataset. The sequence length for generation matches the training settings. Speedup is measured as the ratio of the wall-clock generation throughput when employing Jacobi decoding, and that of the baseline AR decoding.

efficacy of CLLMs using both consistency global loss and consistency local loss. Table 6 demonstrates that the global loss is more efficacious in the training of CLLMs.

4.4. Limitations and Discussion

In our experiments, we observe that achieving significant speedup while maintaining good generation quality with a CLLM relies strongly on having a high-quality Jacobi trajectory dataset. Therefore, data cleaning is crucial, as discussed in Section 3.2.1. Dataset size also plays a role as described in Section 4.3 and shown in Table 4, although to a lesser extent. For instance, Jacobi trajectories generated with only 10% of the Code-Search-Net Python dataset is able to yield a 2.9× speedup as demonstrated in Table 2. However, for open-domain datasets like ShareGPT, more data is necessary for improved efficiency. The computa-

Table 4. Comparison the performance of CLLMs trained with different sizes of Jacobi trajectory datasets on ShareGPT.

TRAJECTORY COUNT	MT-BENCH	INFERENCE SPEEDUP (VARYING LENGTHS)				
		16	32	64	128	256
20K	6.1	1.7×	1.8×	1.4×	1.2×	1.1×
100K	6.4	2.5×	2.4×	2.1×	2.0×	1.5×
500K	6.4	2.7×	2.7×	2.2×	2.1×	1.8×

Table 5. CLLMs' performance versus the fine-tuned baseline on language modeling tasks.

Methods	Speed (tokens/s)	Speedup	PPL (↓)
raw-WikTtext2			
fine-tuned LLaMA2-7B			
+ AR	41.2	1.0×	8.0
+ Jacobi	36.9	1.0×	8.0
+ lookahead	58.1	1.6×	8.0
CLLM-LLaMA2-7B			
+ AR	40.1	1.0×	9.5
+ Jacobi	83.2	2.1×	9.5
+ lookahead	89.5	2.2×	9.5
PTB			
fine-tuned LLaMA2-7B			
+ AR	43.8	1.0×	15.6
+ Jacobi	41.8	1.0×	15.6
+ lookahead	62.0	1.5×	15.6
CLLM-LLaMA2-7B			
+ AR	43.6	1.0×	15.3
+ Jacobi	98.1	2.3×	15.3
+ lookahead	101.5	2.3×	15.3

tion cost for CLLMs training is moderate and discussed in Appendix D.

In our proposed method and experiments, we primarily use output sequences from the teacher (Kim & Rush, 2016) to collect Jacobi trajectories and train a CLLM. This introduces some additional overhead in comparison with conventional model training. On-policy GKD proposed in Agarwal et al. (2023) suggests LLM distillation using a mixture of teacher and student samples or even student samples by themselves can yield high-performance models. One mitigation is therefore to use n-token sequences generated by the trained model itself as the training samples. This can remove the Jacobi trajectory collection overhead, making our proposed method potentially feasible for pre-training.

Results from our language modeling experiments, as detailed in Table 5, demonstrate the robustness of the CLLM when trained on pre-training jobs with a notable speedup. By incorporating on-policy GKD, it is conceivable that a modified version of our proposed method could be employed for LLM pre-training. This modification would equip the pre-trained model with both a strong language modeling capability, as existing models possess, and a high generation speed when employing Jacobi decoding for inference. We leave the opportunities of adapting CLLMs to pre-trained jobs for future work.

Table 6. Comparison the performance of CLLMs trained with different loss design. All models are trained on GSM8K.

LOSS	SPEEDUP	ACCURACY
$\mathcal{L}_{\text{CTG}} + \mathcal{L}_{\text{AR}}$	3.2×	51.3
$\mathcal{L}_{\text{CTG}} + 10 \cdot \mathcal{L}_{\text{AR}}$	3.0×	56.4
$\mathcal{L}_{\text{CTL}} + \mathcal{L}_{\text{AR}}$	2.8×	55.2
$\mathcal{L}_{\text{CTL}} + 10 \cdot \mathcal{L}_{\text{AR}}$	2.4×	56.0

5. Conclusion

In this work, we introduce CLLMs, a new family of LLMs that excel in efficient parallel decoding, designed to significantly enhance the efficiency of Jacobi decoding. Unlike other existing techniques for efficient LLM inference, which often require either additional architectural components (Cai et al., 2024; Li et al., 2024) or draft models (Leviathan et al., 2023; Zhou et al., 2023b; Liu et al., 2023), CLLMs are directly adapted from a target pre-trained LLM. This reduces the complexity associated with additional architecture designs or managing two different models in a single system. In addition, CLLMs can also be integrated seamlessly with other techniques for efficient LLM inference (Dao, 2023; Fu et al., 2024; Ainslie et al., 2023) to achieve greater speedup. We have demonstrated the efficacy of CLLMs on both specific and open domains, revealing a significant improvement in generation speed while preserving generation quality.

Acknowledgments

This work was supported by Key R&D Program of Shandong Province, China (2023CXGC010112), NSF of China (Nos. 62306176, 62102257), National Key R&D Program of China (2022YFB4500200), Natural Science Foundation of Shanghai (No. 23ZR1428700), and CCF-Baichuan-Ebtech Foundation Model Fund.

Impact Statement

This work presents a challenge in machine learning and proposes a solution, the potential negative consequences are not apparent. While it is theoretically possible for any technique to be misused, the likelihood of such misuse occurring at the current stage is low.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Agarwal, R., Vieillard, N., Stanczyk, P., Ramos, S., Geist, M., and Bachem, O. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *arXiv preprint arXiv:2306.13649*, 2023.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Ashkboos, S., Croci, M. L., do Nascimento, M. G., Hoefler, T., and Hensman, J. Slicegpt: Compress large language models by deleting rows and columns, 2024.

Bi, X., Chen, D., Chen, G., Chen, S., Dai, D., Deng, C., Ding, H., Dong, K., Du, Q., Fu, Z., et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chern, E., Zou, H., Li, X., Hu, J., Feng, K., Li, J., and Liu, P. Generative ai for math: Abel. URL https://github.com/GAIR-NLP/abel.

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. *arXiv preprint arXiv:2110.14168*, 2021.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

Gu, Y., Dong, L., Wei, F., and Huang, M. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Kim, Y. and Rush, A. M. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*, 2016.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: Speculative sampling requires rethinking feature uncertainty, 2024.

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., and Han, S. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*, 2023.

Liu, X., Hu, L., Bailis, P., Stoica, I., Deng, Z., Cheung, A., and Zhang, H. Online speculative decoding, 2023.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Ortega, J. M. and Rheinboldt, W. C. *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.

Pan, H., Wang, C., Qiu, M., Zhang, Y., Li, Y., and Huang, J. Meta-kd: A meta knowledge distillation framework for language model compression across domains. *arXiv preprint arXiv:2012.01266*, 2020.

Polišenská, K., Chiat, S., and Roy, P. Sentence repetition: What does the task measure? *International Journal of Language & Communication Disorders*, 50(1):106–118, 2015.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

Santilli, A., Severino, S., Postolache, E., Maiorca, V., Mancusi, M., Marin, R., and Rodolà, E. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*, 2023.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Smadja, F. From n-grams to collocations: An evaluation of xtract. In *29th Annual Meeting of the Association for Computational Linguistics*, pp. 279–284, 1991.

Song, Y. and Dhariwal, P. Improved techniques for training consistency models. *arXiv preprint arXiv:2310.14189*, 2023.

Song, Y., Meng, C., Liao, R., and Ermon, S. Accelerating feedforward computation via parallel nonlinear equation solving. In *International Conference on Machine Learning*, pp. 9791–9800. PMLR, 2021a.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=PxTIG12RRHS.

Song, Y., Dhariwal, P., Chen, M., and Sutskever, I. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wu, S., Fei, H., Qu, L., Ji, W., and Chua, T.-S. Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*, 2023.

Xia, M., Zhong, Z., and Chen, D. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*, 2022.

Xia, M., Gao, T., Zeng, Z., and Chen, D. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., YU, L., Zhang, S., Ghosh, G., Lewis, M., Zettlemoyer, L., and Levy, O. LIMA: Less is more for alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a.

Zhou, Y., Lyu, K., Rawat, A. S., Menon, A. K., Rostamizadeh, A., Kumar, S., Kagy, J.-F., and Agarwal, R. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2023b.

A. Illustration of Consistency Loss Learning Objectives

In our proposed method described in Section 3.2, we use Jacobi trajectories collected from a target model to train the model with a loss that encourages single-step convergence during Jacobi iterations. This is achieved with either choice of the two consistency loss:

- **Global consistency loss**: directly minimize the distance D between any arbitrary point $\mathbf{y}$ on a Jacobi trajectory and the fixed point $\mathbf{y}^*$ in Equation 4.
- **Local consistency loss**: minimize the distance D between any arbitrary point $\mathbf{y}^{(j)}$ on a Jacobi trajectory with its adjacent state $\mathbf{y}^{(j+1)}$ in Equation 5, which thereby also implicitly minimizes the distance between $\mathbf{y}^{(j+1)}$ and the fixed point $\mathbf{y}^*$.

An illustration further depict the global consistency loss and the local consistency loss in Figure 4 and Figure 5.

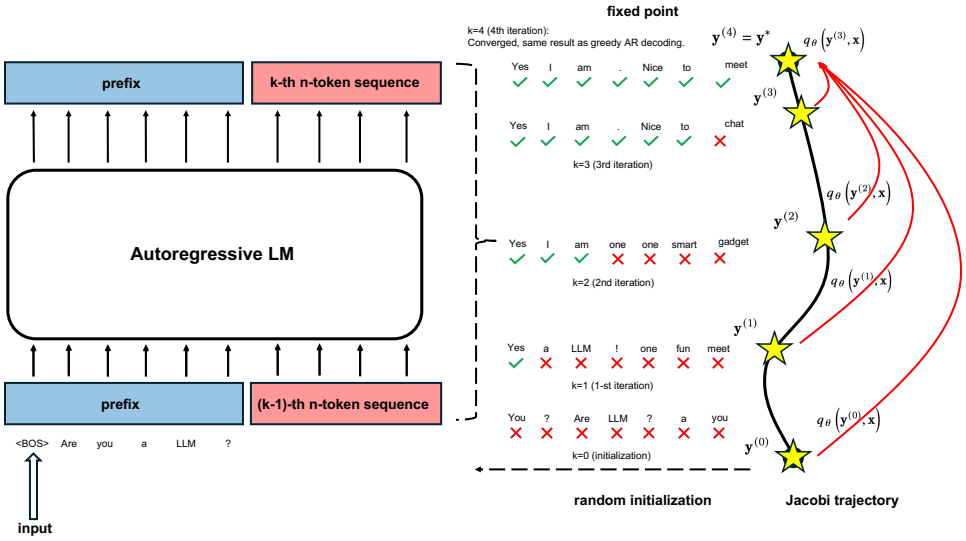

Figure 4. The image illustrates global consistency loss where we aim to directly learn a model q_θ that maps arbitrary n-token sequence $\mathbf{y}^{(0)}$, $\mathbf{y}^{(1)}$, etc.) to the fixed point $\mathbf{y}^*$.

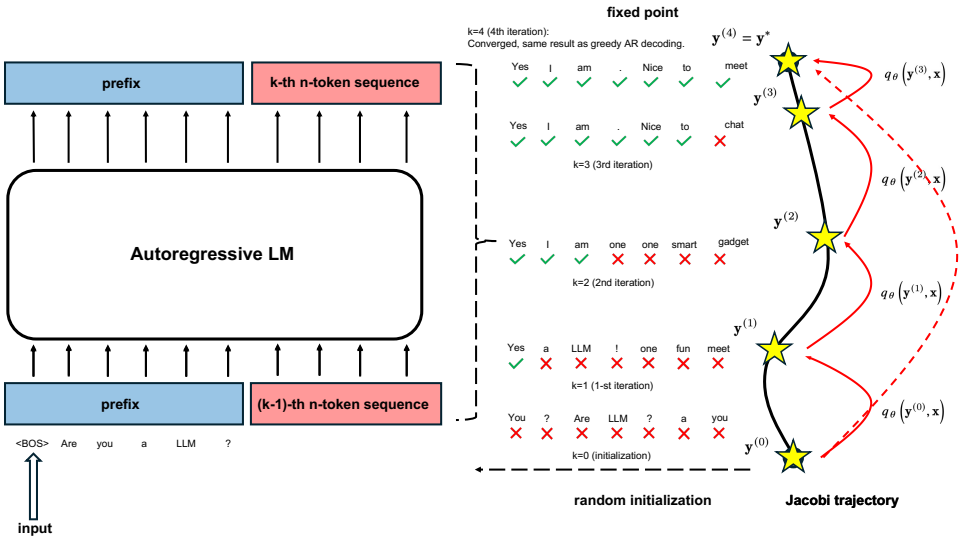

Figure 5. The image illustrates local consistency loss where we aim to learn a model q_θ that maps an arbitrary n-token sequence $\mathbf{y}^{(j)}$ to its next adjacent state, and implicitly mapping the point to the fixed point $\mathbf{y}^*$.

B. Comparison with Baseline Algorithms

In this section, we present a comparative analysis of baseline algorithms for efficient LLM inference. Key features considered are listed below. Table 7 underlines that CLLMs, our proposed method, stands out for its memory efficiency and adaptability, requiring no modifications to the existing model architecture while achieving up to $3.4\times$ inference speedup.

- **Lossless**: whether the method generates exactly the same output distribution as AR decoding does in the backbone model.
- **Training-free**: whether the method requires training.
- **Architecture-design-free**: whether the method requires modifications or adding auxiliary components to pre-trained LLMs (like extra MLP layers, LM heads (Cai et al., 2024), autoregressive heads (Li et al., 2024), etc.).
- **Attention-modification-free**: whether the methods require modifications to exisiting attention mechanism in transformers. For example, this includes tree token verification as appears in Cai et al. (2024).
- **Extra-memory-free**: whether the method requires extra memory conmsumption in the system to accommodate speculative model or extra parameters.
- **Speedup**: Whether the method can effectively deliver inference speedup in practical use cases.

Table 7. All speedups are relative to the vanilla AR. CLLMs has the best memory efficiency and adaptability as it requires no modifications to the model. yes* refers to capability of achieving more than $3\times$ speedup on at least one of our benchmarks. Jacobi decoding doesn't always lead to a speedup as discussed in Section 3.1, so we denote it with yes.

Methods	Lossless	Training-free	Arch-design-free	Attention-mod-free	Extra-memory-free	Speedup
Vanilla AR	yes	yes	yes	yes	yes	no
Jacobi Decoding	yes	yes	yes	yes	yes	yes
Speculative Decoding	yes	yes	yes	yes	no	yes
Lookahead Decoding	yes	yes	yes	yes	no	yes
SD with Distilled Student	yes	no	yes	yes	no	yes
Eagle	yes	no	no	no	no	yes*
Medusa	no	no	no	no	no	yes*
CLLMs (Ours)	no	no	yes	yes	yes	yes*

C. Pesudo Code for Jacobi Decoding with KV Cache

Algorithm 3 Jacobi Decoding with KV Cache

Input: prompt $\boldsymbol{x}$, n-gram size n, past KV cache $\mathcal{K}$, LLM, Jacobi trajectory $\mathcal{J}$
$\boldsymbol{y} \leftarrow$ random tokens from $\boldsymbol{x}$
$n_t \leftarrow 0$ {Initialization of accurate length}
$y_0, \mathcal{K} \leftarrow \text{LLM}(\boldsymbol{x})$ {Prefill phase: generate the first token}
$\boldsymbol{z}_{\text{next}} \leftarrow \text{cat}(y_0, \boldsymbol{y}_{\geq 1})$
repeat
  $\boldsymbol{z}^{\text{current}} \leftarrow \boldsymbol{z}^{\text{next}}$
  $\boldsymbol{z}^{\text{next}}, \mathcal{K} \leftarrow \text{LLM}(\boldsymbol{z}^{\text{current}}, \mathcal{K})$
  $i^* \leftarrow \max\{i \mid \boldsymbol{z}^{\text{current}}_{<i} = \boldsymbol{z}^{\text{next}}_{<i},\ i \in \{0, \ldots, \text{len}(\boldsymbol{z}^{\text{current}}) - 1\}\}$ {Fast-forwarded token count}
  $\boldsymbol{y}_{n_t \leq i' < n_t + i^*} \leftarrow \boldsymbol{z}^{\text{next}}_{<i^*}$ {i' denotes a dummy variable}
  $n_t \leftarrow n_t + i^*$
  Append $\text{cat}\big(\boldsymbol{y}_{<n_t}, \boldsymbol{z}^{\text{next}}_{\geq i^*}\big)$ to $\mathcal{J}$
  Remove KV cache of false tokens from $\mathcal{K}$
  $\boldsymbol{z}^{\text{next}} \leftarrow \boldsymbol{z}^{\text{next}}_{\geq i^*}$
until $n_t = n$
Ouput: $\mathcal{J}$ and $\boldsymbol{y}$

D. Computation cost for CLLMs training

For the computation required for dataset generation, the cost is low and it's a one-time overhead. In the cases where the dataset size is large, for example for CodeSearchNet-Python, only 10% of the dataset is required to generate Jacobi trajectories and the trained CLLMs obtain around 2.5× speedup on average. More details are shown in the table below.

Table 8. Computation required for dataset generation. The estimated generation time is based on sequential generation with batch size = 1. We can further reduce the generation time with serving systems like vLLM (Kwon et al., 2023) with batch size = 16 or more. We give an example of estimated training time with vLLM using batch size = 16 in the table as well. All time is estimated by a single A100 40G GPU*hours.

Dataset	♯ Generated tokens	Estimated generation time	Estimated generation time(vLLM)
Spider	2M	5	< 1
GSM8K	10M	14	~ 1
CodeSearchNet-Python	100M	100	8
ShareGPT	200M	120	10

For the computation required for consistency training, we conclude the time and resources required for training a CLLM in the table below. For the percentage of pre-training cost, we estimate it by $\frac{\text{♯tokens required for training a CLLM}}{\text{♯tokens required for pre-training}}$, where ♯tokens required for pre-training is 1T for LLaMA-7B (Touvron et al., 2023a).

Table 9. Computation required for consistency training.

Dataset	Training time	% of pre-training cost	Training resources
Spider	2 hours	$< 0.01\%$	8 A100 40GB GPUs
GSM8K	12 hours	$\sim 0.01\%$	8 A100 40GB GPUs
CodeSearchNet-Python	22 hours	$\sim 0.1\%$	8 A100 40GB GPUs
ShareGPT	30 hours	~ 0.2 %	8 A100 40GB GPUs