Lossless Acceleration of Large Language Model via Adaptive N-gram Parallel Decoding

Jie Ou, Yueming Chen, Wenhong Tian[*]
University of Electronic Science and Technology of China, Chengdu, China
oujieww6@gmail.com, yuemingchen121@gmail.com
tian_wenhong@uestc.edu.cn

Abstract

While Large Language Models (LLMs) have shown remarkable abilities, they are hindered by significant resource consumption and considerable latency due to autoregressive processing. In this study, we introduce Adaptive N-gram Parallel Decoding (ANPD), an innovative and lossless approach that accelerates inference by allowing the simultaneous generation of multiple tokens. ANPD incorporates a two-stage approach: it begins with a rapid drafting phase that employs an N-gram module, which adapts based on the current interactive context, followed by a verification phase, during which the original LLM assesses and confirms the proposed tokens. Consequently, ANPD preserves the integrity of the LLM's original output while enhancing processing speed. We further leverage a multi-level architecture for the N-gram module to enhance the precision of the initial draft, consequently reducing inference latency. ANPD eliminates the need for retraining or extra GPU memory, making it an efficient and plug-and-play enhancement. In our experiments, models such as LLaMA and its fine-tuned variants have shown speed improvements up to $3.67\times$, validating the effectiveness of our proposed ANPD.

1 Introduction

The advent of Large Language Models (LLMs) such as GPT-4 (OpenAI, 2023), ChatGPT (Brown et al., 2020), LLaMA (Touvron et al., 2023a), and PaLM (Chowdhery et al., 2023), has revolutionized the landscape of natural language processing. However, the majority of LLMs (Touvron et al., 2023a; Anil et al., 2023; Bai et al., 2023) rely on the decoder-only Transformers architecture (Alec et al., 2018), which is intrinsically autoregressive and consequently leads to increased generation time during inference. This characteristic has made the improvement of LLM inference efficiency a significant research area within the natural language processing community.

Model compression techniques such as quantization (Han et al., 2015), pruning (Molchanov et al., 2016), and distillation (Hinton et al., 2015) have been employed to alleviate the computational costs associated with LLMs. Recently, innovative methods such as early exit strategies (Yang et al., 2023b; Bae et al., 2023; Kong et al., 2022; Schuster et al., 2022; Varshney et al., 2023) and speculative decoding (Kim et al., 2023; Xia et al., 2022; Leviathan et al., 2023; Spector and Re, 2023; Zhang et al., 2023a) have been proposed to speed up the inference process. While these methods are effective, they typically necessitate modifications to the model architecture and re-training, which can incur substantial costs. Additionally, they may alter the model's output and require extra GPU memory needs. A method avoiding draft models using retrieval is presented in (He et al., 2023), but it requires a large database.

For certain LLMs, such as LLaMA, the tokenization process can dissect a single word into multiple tokens, thereby exacerbating inference latency. As illustrated in Figure 1, the token count exceeds the word count, resulting in an increased number of autoregressive generation steps. In such scenarios, given the constraints imposed by contextual information, the search space for predicting the next token that forms part of a word based on the current token is significantly narrowed. Moreover, contextual information can often be leveraged to identify patterns and correlations between words. This is especially evident for simple phrases and paragraphs, where the context can provide clear indicators that reduce the dependency on LLM decoding.

Based on the above motivation, this paper presents a novel approach, the **A**daptive **N**-gram **P**arallel **D**ecoding (**ANPD**), designed to enhance inference efficiency without necessitating retrain-

[*]Corresponding author

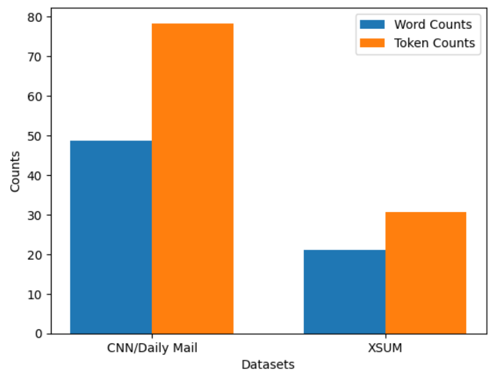


Figure 1: The comparative analysis of the number of words and tokens after tokenizer processing for the CNN/Daily Mail and XSUM datasets.

ing or the integration of an auxiliary small language model. ANPD dynamically generates draft outputs via an adaptive N-gram module using real-time statistics, after which the drafts are verified by the LLM. This characteristic is exactly the difference between ANPD and the previous speculative decoding methods. The primary contributions of this work can be summarized as follows:

- We propose ANPD, a novel and lossless algorithm that offers a plug-and-play module for acceleration of LLM inference.
- We propose an adaptive N-gram modeling strategy that is specifically adapted for LLMs, markedly diminishing the complexity of language modeling and reducing the dependency on large-scale textual datasets.
- We propose a Multi-Level N-gram (MLN) algorithm aimed at increasing the precision of draft outputs, thereby enhancing the efficiency of the acceleration process.
- We conduct extensive experiments on various models and datasets, demonstrating the robust acceleration capabilities of ANPD, with a notable increase of 1.95×-3.67× on LLaMA and its fine-tuned derivatives.

2 Related Work

Inference systems. The development of specialized inference systems for Large Language Models (LLMs), such as NVIDIA's TensorRT-LLM (NVIDIA, 2023), Orca (Yu et al., 2022), FlexGen (Sheng et al., 2023), and DeepSpeed Inference (Aminabadi et al., 2022), represents a notable advancement in the field. Despite progress, there is still a gap in the careful co-design of algorithms and systems, which is necessary to fully harness the potential of the hardware.

Compression. Efficient LLM inference is facilitated by techniques such as quantization (Han et al., 2015; Frantar et al., 2022; Dettmers et al., 2022; Xiao et al., 2023), pruning (Bansal et al., 2023; Frantar and Alistarh, 2023; Liu et al., 2023), distillation (Tang et al., 2019; Touvron et al., 2021), and exit early strategies (Schuster et al., 2022; Kong et al., 2022; Yang et al., 2023b; Bae et al., 2023; Del Corro et al., 2023) suggest that some tokens can be accurately generated using only a fraction of the model layers. Token Prunings (Hou et al., 2022; Yao et al., 2022; Zhang et al., 2023b) reduce memory and computational demand to accelerate the inference process by prioritizing crucial tokens. These methods enhance efficiency but may necessitate model alterations, re-training, and potentially reduce accuracy.

Speculative Execution. Speculative execution (Burton, 1985), adapted as speculative decoding in LLMs (Chen et al., 2023; Leviathan et al., 2023), has improved inference speeds by preempting computations. SpecInfer (Miao et al., 2023) leverages existing distilled, quantized, and pruned variants of an LLM, to build a small speculative model pool to guide speculation. However, these approaches require a high-quality draft model, and increase the memory footprint. Leviathan et al. (2023) also mentioned that unigram and bigram can be used as draft models, but they did not propose a method on how to build a bigram model for the actual running LLMs. Yang et al. (2023a) presented a method of copying reference tokens to the decoder, though its utility is limited by a dependency on repeated text. These techniques increase resource use and compel specialized training, such as distillation, for the draft model to ensure compatibility with the primary model.

3 Method

Figure 2 illustrates the framework and workflow of proposed ANPD. We explain the original autoregressive decoding in the Appendix A.1.

3.1 Adaptive N-gram Parallel Decoding

Figure 2 illustrates the pipeline of our ANPD. The process begins with tokenizing the input text into tokens. The N-gram module's Memory actually stores token ids to streamline processing, Figure 2

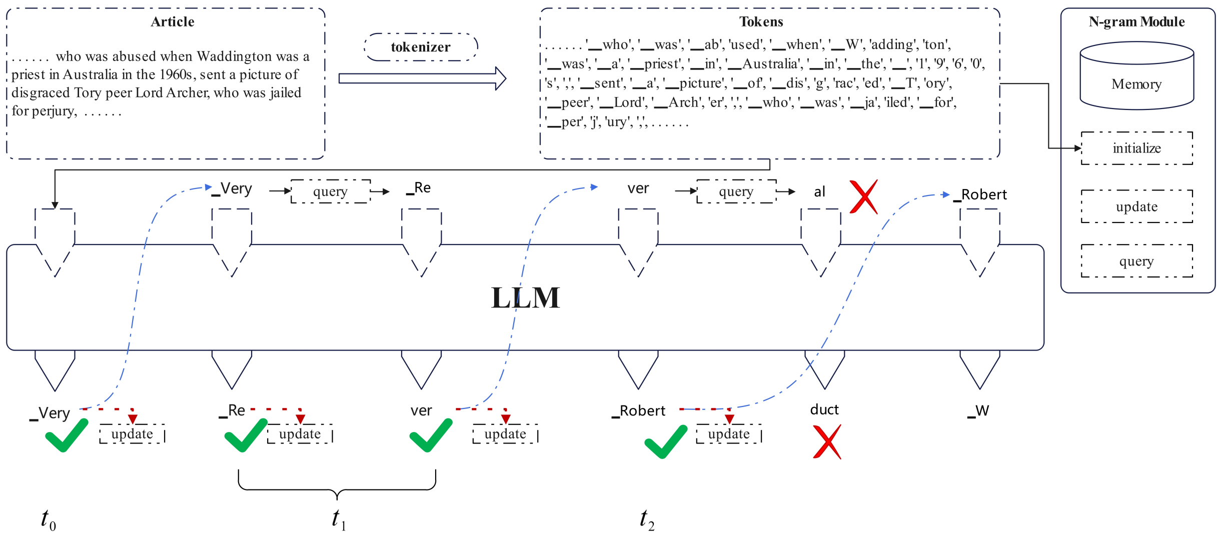

Article
...... who was abused when Waddington was a priest in Australia in the 1960s, sent a picture of disgraced Tory peer Lord Archer, who was jailed for perjury, ...
tokenizer
Tokens
...... '_who', '_was', '_ab', 'used', '_when', '_W', 'adding', 'ton', '_was', '_a', '_priest', '_in', '_Australia', '_in', '_the', '_', '1', '9', '6', '0', 's', ',', '_sent', '_a', '_picture', '_of', '_dis', 'g', 'rac', 'ed', '_T', 'ory', '_peer', '_Lord', '_Arch', 'er', ',', '_who', '_was', '_ja', 'iled', '_for', '_per', 'j', 'ury', ',', ...
N-gram Module
Memory
initialize
update
query
_Very
query
_Re
ver
query
al
_Robert
LLM
_Very
update
_Re
update
ver
update
_Robert
update
duct
_W
t_0
t_1
t_2

Figure 2: The pipeline of the ANPD. The tokenizer first processes the text to obtain a list of tokens. These tokens are used to initialize the N-gram module. Simultaneously, these tokens are fed into the LLM for processing via autoregression. The predicted token at time t_0 in the figure is "_Very". This word is used as a query into the N-gram module, yielding the token "_Re", which along with the "_Very" are sent to the LLM for inference at time t_1. A green checkmark signifies acceptance of the predicted token, while a red cross indicates rejection. Each accepted token, is combined with the first $N-1$ tokens to form a tuple, and the **update** method is called to refresh the N-gram module.

shows tokens as the basis for modeling to make it easier for readers to understand and improve readability. Next, the LLM engages in autoregressive inference, divided into two parts: 1. Prefill, where the full prompt is input to generate the first token; 2. Decoding, ANPD feeds multiple tokens from the N-gram module into the LLM, and the LLM uses kv-cache for efficient computations to validate tokens for parallel output generation. Tokens that fail validation are discarded along with subsequent tokens. Simultaneously, we use an adaptive strategy to update the N-gram module throughout LLM generation, avoiding reliance on static Memory.

Token Level N-gram Module. Contextual information is vital for content extraction, summarization, and code generation, as it helps refine the search space during each LLM decoding step. This includes strong correlations among tokens within words and between words in phrases and contexts. We constructed a token-level N-gram module to uniformly model the above correlations. The N-gram module[1] is a probabilistic language model, that predicts the next item in a sequence using an $(N-1)$-th order Markov model, where N is the subsequence length. For a token sequence $x_1, x_2, ..., x_{t-1}$, the model estimates the probability of x_t based on the preceding $N-1$ tokens, as $P(x_t|x_1,...,x_{t-1}) \approx P(x_t|x_{t-N+1},...,x_{t-1})$. In a bigram model ($N=2$), the sentence probability is:

$$P(x_1, x_2, ..., x_n) \approx \prod_{i=2}^{n} P(x_i|x_{i-1}), \qquad (1)$$

probabilities $P(x_i|x_{i-1})$ derive from frequency counts in the corpus. We have architected the N-gram module to encapsulate three principal functions essential for its operation:

- **Initialize:** using a tokenizer converts each prompt into a sequence of token ids. It then performs probabilistic statistics on these ids and records the probability for each token tuple.
- **Update:** during the decoding, each new token is paired with the previous $N-1$ tokens to form a tuple, used to update the module's probability Memory.
- **Query:** the query operation utilizes the token ids tuple, constructed through the subsequence from $t-N+1$ to $t-1$, to predict the next token x_t, effectively leveraging the statistical results established by the preceding functions.

[1]https://web.stanford.edu/~jurafsky/slp3/3.pdf

These functions collectively enable the N-gram module to dynamically adapt to the evolving text generation process, ensuring that each token generated is contextually relevant and statistically coherent.

Parallel Decoding. The parallel decoding in our ANPD is similar to the speculative decoding approach and occurs in two distinct stages:

1. Drafting: the N-gram module is harnessed to generate a sequence of subsequent tokens. By iterating through K steps, the module constructs a preliminary draft tokens with length K. Specifically, the draft module generates a series of K temporary tokens $x_{i+1}, ..., x_{i+K}$, succeeding a given prompt sequence $x_1, ..., x_i$.

2. Verification: the original Large Language Model (LLM) verifies the proposed draft tokens, through a singular forward pass as $P(x^{'}_{i+K+1}|(k,v)_1, ..., (k,v)_i, x_{i+1}, ..., x_{i+K})$, within which the LLM computes the probability distributions for each draft token, then to ascertain their congruence with the proposed draft tokens $x_{i+1}, ..., x_{i+K}$. If a draft token x_j does not pass this validation, it is replaced by the LLM's prediction $x^{'}_j$, and a new drafting begins from this token.

The ANPD enhances efficiency by eliminating the need for a smaller draft deep learning model, leveraging the much lower computational cost N-gram module to accelerate LLM inference. For LLMs, conducting parallel inference of K tokens introduces a negligible increase in computational latency compared to single token autoregressive inference, as shown in Figure 7 in Appendix A.2. Meanwhile, our technique is intrinsically capable of yielding at least j tokens ($1 \leq j \leq K+1$) for each decoding step, this intrinsic capability fundamentally assures, in principle, an acceleration of the decoding processes within the Large Language Model (LLM), thereby enhancing the overall computational throughput and reducing latency. The implementation of the two-stage process confers upon the ANPD the ability to iteratively refine draft outputs. Furthermore, this guarantees that our ANPD method is lossless, maintaining consistency with the original LLM's generated content. The detailed procedure of ANPD is presented in Algorithm 1, with a comprehensive explanation available in Appendix A.3.

Algorithm 1 Adaptive N-gram Parallel Decoding

```
1: Input: $prompt$, $K$, $M$
2: Output: $O$
3: $token\_ids \leftarrow$ TOKENIZER($prompt$)
4: $Memory \leftarrow$ INITIALIZE($token\_ids$)
5: $O \leftarrow [\,]$, $drafts \leftarrow [\,]$
6: $pred \leftarrow$ LLM($prompt$)
7: $drafts.append(pred[-1])$
8: while $length(O) < M$ do
9:     $token\_ids.append(drafts[1])$
10:    $O.append(token\_ids[-1])$, UPDATE($O[-1]$)
11:    $tmp\_token\_ids \leftarrow token\_ids[-N+1:]$
12:    for $k \leftarrow 1$ to $K$ do
13:        $tmp \leftarrow tmp\_token\_ids[-N+k:]$
14:        $drafts.append($QUERY$(tmp))$
15:        $tmp\_token\_ids.append(drafts[-1])$
16:    end for
17:    $predicts \leftarrow$ LLM($drafts$)
18:    for $j \leftarrow 2$ to LENGTH($drafts$) do
19:        if $drafts[j] == predicts[j-1]$ then
20:            $O.append(drafts[j])$
21:            UPDATE($drafts[j]$)
22:            $token\_ids.append(drafts[j])$
23:        else
24:            break
25:        end if
26:    end for
27:    if $j ==$ LENGTH($drafts$) then
28:        $drafts \leftarrow [predicts[j]]$
29:    else
30:        $drafts \leftarrow [predicts[j-1]]$
31:    end if
32: end while
```

3.2 Multi-Level N-gram

The predictive accuracy of the N-gram module is known to correlate with N, larger N values generally result in more accurate content predictions. This effect is especially noticeable in settings with the longer context of Language Model (LM) tasks, where increasing N can markedly decrease the frequency of prediction errors.

While a larger N tends to improve the predictive accuracy of the N-gram module, it may not always result in a successful match during the Query operation. To address this, we propose the Multi-Level N-gram (MLN) approach, which is based on optimal prefix matching. The MLN design initializes $N-1$ separate modules, each corresponding to an n-gram module ($n \in [2, N]$). During prediction,

Algorithm 2 Multi-Level N-gram

```
1:  Input: $tmp$, $N$,$token\_ids$
2:  Output: $result$
3:  $Memory \leftarrow$ INITIALIZE($token\_ids$)
4:  $result \leftarrow$ NULL
5:  $n \leftarrow N$
6:  while $n \geq 2$ do
7:      $pred \leftarrow$ QUERY($query, n$)
8:      if $pred \neq$ NULL then
9:          $result \leftarrow pred$
10:         break
11:     end if
12:     $n \leftarrow n - 1$
13: end while
14: return $result$
```

the query starts with the largest N and proceeds to lower n levels, stopping when a successful match is found as shown in Algorithm 2.

4 Experiments

4.1 Implementation Details

We selected a diverse range of models, varying in scale, architectural design, and training approaches, to ensure a thorough evaluation, including LLaMA-7B (Touvron et al., 2023a), LLaMA-2-7B (Touvron et al., 2023b), ChatGLM3-6B (Du et al., 2022), LLaMA-2-13B, CodeLLaMA-7B (Roziere et al., 2023), CodeLLaMA-13B, and instruction-tuned variants such as Alpaca-7B and Alpaca-CNN/DM-7B, fine-tuning details are provided in the Appendix A.4. We use one RTX-3090 GPU for all 7B models, while the larger 13B models necessitate four RTX-3090 GPUs and the accelerate[2] library.

4.2 Datasets & Metrics

To validate the effectiveness of our method in accelerating text generation for LLMs, we concentrated on two tasks: text summarization and code generation, utilizing datasets such as CNN/Daily Mail (CNN/DM) (Hermann et al., 2015), Extreme Summarization (XSum) (Narayan et al., 2018), and the HumanEval (Chen et al., 2021). For additional details on the evaluation settings, please see Appendix A.5. We employ the speed-up ratio as the evaluation metric, which is calculated by dividing the inference time of the autoregressive process by the inference time of the ANPD process, under identical conditions across all samples (For summarization tasks, we use a sample size of 1000 to ensure statistical significance, as recommended by (Zhang et al., 2023a)). This metric intuitively demonstrates the performance improvement in speed when using the ANPD algorithm.

4.3 Main Results

In Table 1, we present a comparative analysis that outlines the acceleration benefits for various models and datasets. We have selected (Zhang et al., 2023a) for comparison. Not only are their experimental datasets and models aligned with ours, but their methodologies are also open-sourced to facilitate easy replication. The prompts used with these models are comprehensively documented in Appendix A.5 to facilitate further examination and ensure the reproducibility of the results reported in this paper.

As illustrated in Table 1, the ANPD algorithm consistently accelerates inference across various models, including the base LLM, the instruction-fine-tuned Alpaca, and the model fine-tuned with dataset-specific instructions, indicating its robustness and efficiency in accelerating text generation. Remarkably, for the LLaMA-7B model, ANPD can speed up the inference speed over $2.0\times$, which is still valid on LLaMA2. Our method achieves a twofold ($2.9088\times$ vs. $1.3293\times$) increase in acceleration compared to (Zhang et al., 2023a) on the LLaMA-2-13B. Despite the ChatGLM3 model having a significantly larger vocabulary (nearly twice that of LLaMA, the token/word ratio will be closer to 1), our ANPD algorithm still achieves a speed-up of $1.7046\times$ and $1.6647\times$ for CNN/DM and XSum, respectively. In ChatGLM3, ANPD's predictive mechanism primarily leverages the associative relationships between phrases and individual words, rather than engaging in token-level predictions within the words themselves. So, ANPD maintains robustness and consistently enhances inference speeds across varied LLMs. Owing to the presence of a high occurrence of correlated patterns in code writing tasks, which significantly enhanced the prediction accuracy of the ANPD algorithm. The ANPD algorithm was able to achieve a substantial speed-up of $3.6665\times$ on the HumanEval, but (Zhang et al., 2023a) only has a speed-up of $1.6758\times$ for CodeLLaMA-13B.

4.4 Ablation Study

We conduct an analysis of hyperparameters on CNN/DM dataset, focusing primarily on K and N. In

[2]https://github.com/huggingface/accelerate

Model	shot	CNN/DM	XSum
LLaMA-7B	1	2.7455x	3.1195x
Alpaca-7B	0	2.5566x	2.3022x
Alpaca-CNN/DM-7B	0	1.9481x	2.0561x
LLaMA-2-13b (Zhang et al., 2023a)	1	1.3293x	1.2801x
LLaMA-2-7B	1	2.8604x	2.7973x
LLaMA-2-13B	1	2.9088x	2.6063x
ChatGLM3-6B	0	1.7046x	1.6647x

Model	shot	HumanEval
CodeLLaMA-13B (Zhang et al., 2023a)	0	1.6758x
CodeLLaMA-7B	0	3.5985x
CodeLLaMA-13B	0	3.6665x

Table 1: The comparison of acceleration effects on different models and datasets.

Figure 3, we set N to 2, and perform a comparative analysis of the parameter K. Our findings indicate that increasing K contributes to a greater acceleration effect, however, the acceleration gains plateau when K lies within the range of 6 to 8.

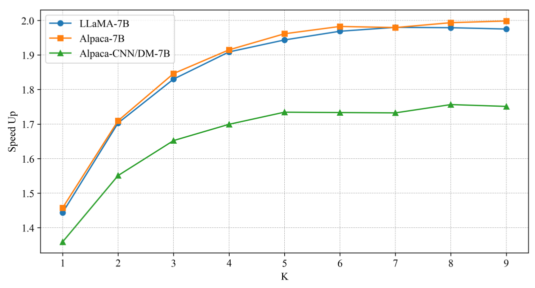


Figure 3: Speed up ratio of LLM for different K.

Based on the experiment in Figure 3, we selected 6, 7, and 8 for K to conduct further hyperparameter combination experiments, as illustrated in Figures 4 and 5. The experimental results indicate that the Multi-Level N-gram (MLN) approach enhances inferential speed as the parameter N increases. However, beyond $N = 5$, further increments in N yield no significant additional gains. Additionally, the effect of the parameter K on acceleration is relatively stable; as shown in Figure 3, the acceleration effect reaches a plateau within the range of 6 to 8 for K. These findings are consistent across different models with different N.

Based on the empirical evidence presented in Figure 4 and Figure 5, a pragmatic choice for N and K can be posited at $N = 5$ and $K = 7$ respectively. The analogous experiments pertaining to the HumanEval dataset have been relegated to Appendix A.6 for reference, similar conclusions can also be observed in this dataset. While employing the Multi-Level N-gram (MLN) has improved the accuracy of draft predictions, we have also carried out distinct experiments (Figure 10, Appendix A.6) using N-gram modules without MLN, to demonstrate that simply enlarging the value of N is not effective.

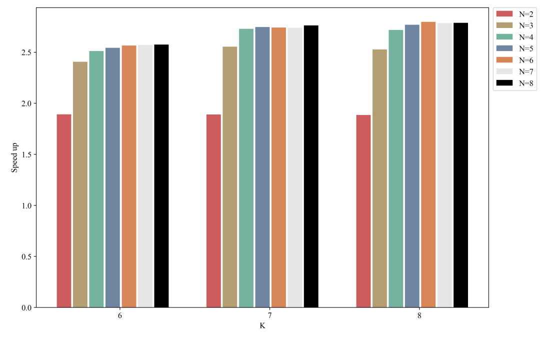


Figure 4: Decoding speed up ratio of LLaMA-7B for different K and N.

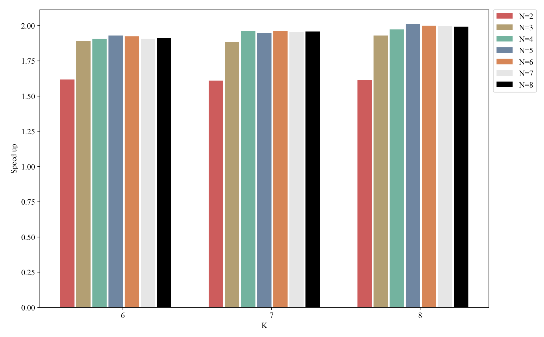
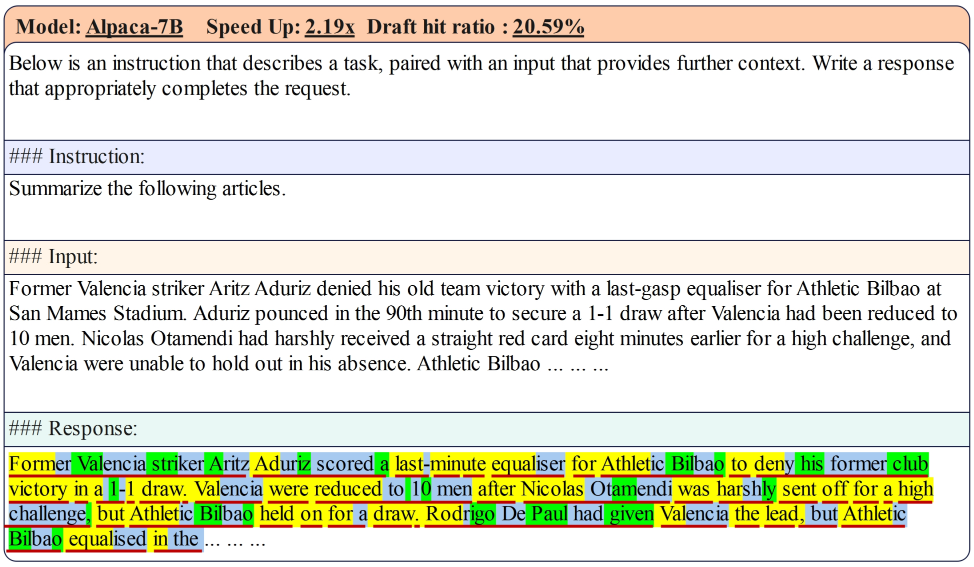


Figure 5: Decoding speed up ratio of Alpaca-CNN/DM-7B for different K and N.

4.5 Case Study

Figure 6 showcases a detailed example of the ANPD inference process, utilizing the Alpaca-7B model on a sample from the CNN/DM test set. The Alpaca-7B model, which has been fine-tuned with instructions, was chosen due to its broad applicability in practical scenarios. In this example, the ANPD algorithm is configured with $N = 5$ and $K = 7$, achieving a 2.19× decoding speed-up compared to the original autoregressive process, with a draft text pass rate (Draft hit ratio, α) of 20.59% in the LLM verification phase. Based on the hit ratio, we can derive the theoretical upper bound of acceleration as $(\alpha \times K) + 1$, we can calculate that the theoretical speed-up is 2.44, as the loss caused by implementation problems will be slightly higher than the actual acceleration rate. The Figure 6 uses red underlines to represent a decoding step, including drafting and verification, with the yellow background indicating the beginning of one step. Light blue and green backgrounds mark the draft



Model: Alpaca-7B Speed Up: 2.19x Draft hit ratio : 20.59%

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

Instruction:

Summarize the following articles.

Input:

Former Valencia striker Aritz Aduriz denied his old team victory with a last-gasp equaliser for Athletic Bilbao at San Mames Stadium. Aduriz pounced in the 90th minute to secure a 1-1 draw after Valencia had been reduced to 10 men. Nicolas Otamendi had harshly received a straight red card eight minutes earlier for a high challenge, and Valencia were unable to hold out in his absence. Athletic Bilbao ... ...

Response:

Former Valencia striker Aritz Aduriz scored a last-minute equaliser for Athletic Bilbao to deny his former club victory in a 1-1 draw. Valencia were reduced to 10 men after Nicolas Otamendi was harshly sent off for a high challenge, but Athletic Bilbao held on for a draw. Rodrigo De Paul had given Valencia the lead, but Athletic Bilbao equalised in the ... ...

Figure 6: Visualizing the step-by-step inference process of ANPD: An example from CNN/DM.

content that has passed verification. This example demonstrates that inference acceleration primarily benefits from the combination of names (e.g., _Athlet, ic, _Bil, ba, o), partial words(e.g., _har, sh, ly), and phrases (e.g., _reduced, _to), aligning with the motivation behind the ANPD algorithm. The ANPD can quickly capture the association between tokens and words based on this information, and establish the prediction model, thus accelerating the end-to-end decoding process.

4.6 User Friendly

As ANPD does not involve additional deep learning models or plug-in databases, it does not require complex initialization processes and environment configuration installations. Consequently, users can employ it directly and with great convenience, as illustrated in Listing 1. We plan to release the associated open-source software packages on GitHub[3], making them accessible for everyone to utilize and contribute to.

Listing 1: Python example

```python
from anpd import anpd_llm
# import other libraries as usual
model = AutoModel.from_pretrain()
model = anpd_llm(model, n=5, k=7)
prompt = "Hello,World!"
result = model.gen(prompt)
```

[3]https://github.com/oujieww/ANPD

5 Conclusion

In this paper, we presented the ANPD algorithm, a novel and lossless approach to accelerate the Large Language Models (LLMs) inference. This algorithm implements an adaptive N-gram modeling strategy, reducing the necessity for large corpora and eliminating the requirement to build an additional deep-learning draft language model. The Multi-Level N-gram (MLN) strategy not only enhances draft output accuracy but also further boosts efficiency. Our empirical studies across various models and datasets validate the ANPD algorithm's effectiveness, with a remarkable peak acceleration of up to 3.67× achieved. The ANPD algorithm has demonstrated its potency as a powerful tool for enhancing the efficiency of LLMs. As a plug-and-play module, it enables more extensive and pragmatic use of LLMs in various real-world contexts.

Future Works. We believe that ANPD can be further enhanced in two key aspects:

1. Incorporating the specific characteristics of individual LLMs (e.g., LLaMA, ChatGLM) by creating features tailored to different LLMs to further accelerate the inference performance.

2. Exploring the possibility of generating multiple tokens in parallel during the LLMs verification process to further accelerate the inference performance.

6 Acknowledgements

This research is supported by the National Key Research and Development Program of China with Grant ID 2018AAA0103203 and the Chengdu Science and Technology Project with Grant ID 2022-YF05-02014-SN. This research is also supported by Huawei MindSpore Team for providing some experimental equipment, technical assistance and experience sharing.

References

Radford Alec, Narasimhan Karthik, Salimans Tim, and S Ilya. 2018. Improving language understanding with unsupervised learning. *Citado*, 17:1–12.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5910–5924.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, Sravan Bodapati, Katrin Kirchhoff, and Dan Roth. 2023. Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

F Warren Burton. 1985. Speculative computation, parallelism, and functional programming. *IEEE Transactions on Computers*, 100(12):1190–1193.

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*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *arXiv preprint arXiv:2307.02628*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Elias Frantar and Dan Alistarh. 2023. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Le Hou, Richard Yuanzhe Pang, Tianyi Zhou, Yuexin Wu, Xinying Song, Xiaodan Song, and Denny Zhou. 2022. Token dropping for efficient bert pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3774–3784.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. 2023. Speculative decoding with big little decoder. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jun Kong, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. 2022. Accelerating inference for pretrained language models by unified multi-perspective early exiting. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4677–4686.

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.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745.

NVIDIA. 2023. Tensorrt-llm: NVIDIA tensorrt for large language models.

OpenAI. 2023. Gpt-4 technical report.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Benjamin Frederick Spector and Christopher Re. 2023. Accelerating llm inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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*.

Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. 2023. Accelerating llama inference by enabling intermediate layer decoding via instruction tuning with lite.

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

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

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

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2023b. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Zhewei Yao, Xiaoxia Wu, Conglong Li, Connor Holmes, Minjia Zhang, Cheng Li, and Yuxiong He. 2022. Random-ltd: Random and layerwise token dropping brings efficient training for large-scale transformers. *arXiv preprint arXiv:2211.11586*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023a. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv e-prints*, pages arXiv–2205.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2023b. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*.

A Appendix

A.1 Standard Autoregressive Decoding

Transformer-based LLMs use autoregressive decoding, taking text input $(x_1, ..., x_{t-1})$ to predict the next token probability, $p(x_t|x_1, ..., x_{t-1})$. Efficiency is improved by caching past states as $p(x_t|(k, v)_1, ..., (k, v)_{t-1})$. This is an autoregressive process, LLM can only predict one token at a time, as subsequent tokens are dependent on the previous token.

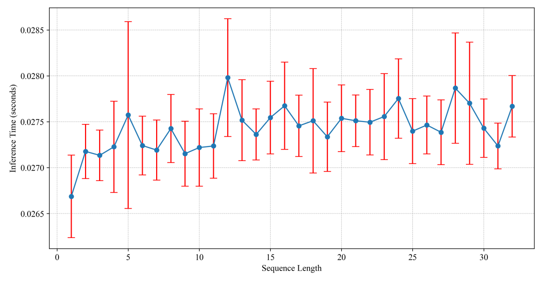


Figure 7: A single decoding step latency of LLaMA-7B is recorded with different K.

A.2 Parallel Decoding Analysis

Figure 7 evaluates the latency impact of processing varying numbers of tokens in the parallel decoding step while maintaining a constant prompt size of 512 tokens in the key-value (KV) cache. The results indicate that small increments in K do not significantly affect latency. It provides the opportunity to verify multiple draft tokens simultaneously without incurring significant additional latency.

A.3 Algorithm Details

In Algorithm 1, the complete process of our ANPD is demonstrated. The variable K denotes the length of the draft output (draft steps), M signifies the maximum length for LLM generation, and O is an output list utilized for recording the token ids of the generated tokens. The presented algorithm initiates by utilizing a prompt to generate token ids, which are then stored in the N-gram module Memory. As delineated in line 6 of the pseudocode, the LLM engages in the prefill phase to produce a valid token prediction (*pred*). This token is essential for updating the output O, the Memory, and the draft array $drafts$. The decoding initiates with the slicing of the most recent $N-1$ tokens from the complete token ids (*token_ids*), these tokens are then utilized as the input for the $QUERY$ in the decoding loop, which spans from line 8 to the terminal line of the algorithm. Throughout the draft generation phase, the tokens within the draft are dynamically updated by $QUERY$. Subsequently, at line 17, parallel decoding is applied to the drafts. This is followed by a meticulous comparison of each token in the draft against the predictions rendered by the large language model (LLM) to ensure alignment and consistency. The comparison process is halted upon the detection of a divergence at the j^{th} draft token. At this critical point, the procedure reverts to the next token of the last consistent token provided by the large language model (LLM) to commence a new draft iteration. If the entire content of the draft withstands verification, the final token predicted by the LLM is then adopted to initiate the generation of a new draft sequence.

A.4 Alpaca Train Details

We train the Alpaca-7B model followed by (Taori et al., 2023). The training dataset employed consists of approximately 52,000 instances, as introduced in (Taori et al., 2023). For fine-tuning the LLaMA-7b model, the learning rate was set to 2×10^{-5}, with a batch size of 128, across a total of 3 epochs. To facilitate effective training within the computational constraints, the $gradient\_accumulation\_steps$ parameter was used. We used float16 for training, engaging the $stage2$ optimization of DeepSpeed and enabling $gradient\_checkpointing$ on one NVIDIA-A100 GPU.

In the case of Alpaca-CNN/DM-7B, we random sample a subset of 30,000 data samples from the CNN/DM trainset, following the alpaca template provided by (Taori et al., 2023), as shown in Figure 8. Notably, the remaining training hyperparameters are the same as Alpaca-7B, except the number of epochs is 5.

```
Below is an instruction that
 ↪  describes a task, paired with
 ↪  an input that provides
 ↪  further context. Write a
 ↪  response that appropriately
 ↪  completes the request.

### Instruction:
{instruction}

### Input:
{input}

### Response:
```

Figure 8: Alpaca template, the instruction is "Summarize the following articles." in our experiments.

A.5 Evaluation

Our evaluation involved a 1-shot setup for non-instruction tuned models and a 0-shot setting for instruction-tuned models, both using ROUGE-2 scores to assess text summarization. For code generation, a 0-shot setting with pass@1 metrics was employed. It is important to note that our approach does not modify the fundamental output or computational processes of existing Large Language Models (LLMs), thereby preserving their inherent performance capabilities. Therefore, we do not conduct a detailed analysis of the accuracy in this paper. For the 0-shot setting, the alpaca template illustrated in Figure 8 is utilized for the summarization task. For the 1-shot setting, the input template employed is depicted in Figure 9. Regarding the use of CodeLLaMA for HuamnEval, we directly enter the text corresponding to the prompt keyword of the sample content, and corresponding instructions have been written for each sample.

```
Article: {shot_article}
Summary: {shot_summary}
Article: {article}
Summary:
```

Figure 9: 1-shot Template.

Our proposed ANPD maintains the integrity of the original model's predictive performance. As delineated in Tables 2 and 3, we report the empirical evaluation results on the widely-adopted benchmarks CNN/DM and HumanEval, respectively. Notwithstanding minor discrepancies in the findings, these can be ascribed to a documented caching anomaly in the issue[4]; nonetheless, their influence on the overall efficacy of ANPD is negligible.

Method	shot	ANPD	CNN/DM
LLaMA-7B	1		8.66
LLaMA-7B	1	✓	8.64
Alpaca-7B	0		10.84
Alpaca-7B	0	✓	10.83
Alpaca-CNN/DM-7B	0		17.16
Alpaca-CNN/DM-7B	0	✓	17.23
LLaMA-2-13B	1		10.58
LLaMA-2-13B	1	✓	10.61
ChatGLM3-6B	0		14.60
ChatGLM3-6B	0	✓	14.54

Table 2: The comparison of the ROUGE-2 for CNN/DM.

A.6 Multi-Level N-gram

In the experiment shown in Figure 10, where the Multi-Level N-gram (MLN) strategy was not uti-

[4]https://github.com/huggingface/transformers/issues/25420

Method	shot	ANPD	HumanEval
CodeLLaMA-7B	0		0.3109
CodeLLaMA-7B	0	✓	0.3109
CodeLLaMA-13B	0		0.3415
CodeLLaMA-13B	0	✓	0.3415

Table 3: The comparison of the Pass@1 for HumanEval.

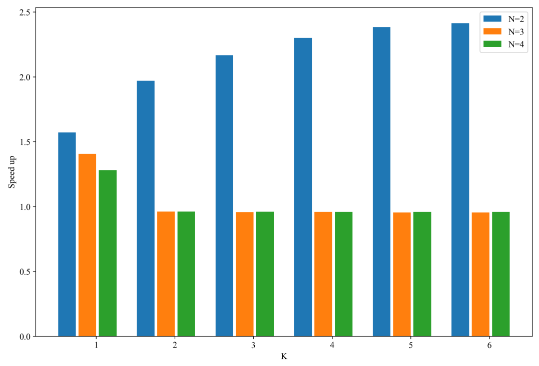


Figure 10: The acceleration comparison of the ANPD for different K and N, without MLN, using the CodeLLaMA-7B.

lized, we reverted to testing the original N-gram module. The results from this setting indicate that merely increasing the N value—referring to the length of the word sequences considered by the model—does not lead to a faster inference process in LLMs. This is primarily attributed to the fact that a larger N value results in fewer successful matches during the Query phase. As the N-gram sequences become longer, the likelihood of finding an exact match in the database diminishes, which in turn negates the potential gains in inference speed from expanding the N-gram size.

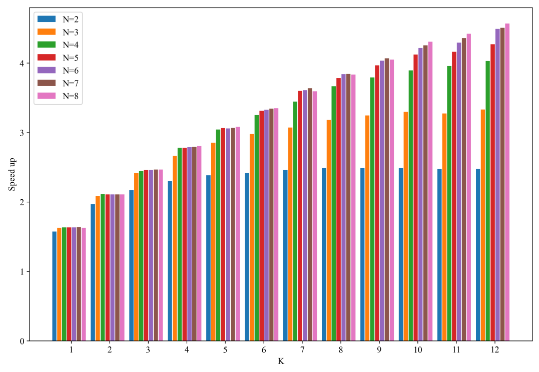


Figure 11: The acceleration comparison of the ANPD for different K and N, with MLN, using the CodeLLaMA-7B.

Figure 11 Experiments on hyperparameters K and N using the CodeLLaMA model on HumanEval. Empirical analyses suggest that the setting, in which the N-gram length (N) is set to 5 and the number of top candidates (K) is set to 7, leads to a marked improvement in performance. This specific configuration yields an inference acceleration close to 3.6 $\times$ faster than the baseline. Furthermore, with a smaller N, as K increases, the acceleration effect tends to reach convergence more quickly.

A.7 More Models

We also conducted relevant experiments on the original OPT model (Zhang et al., 2022) and instruction-tuned Alpaca-OPT-6.7B download from the huggingface[5]. The experimental results in Table 4 further verify that the ANPD we proposed has good robustness and can effectively accelerate inference for different models.

Model	shot	CNN/DM	XSum
OPT-6.7B	1	3.0948x	3.3672x
Alpaca-OPT-6.7B	0	3.0249x	3.1442x

Table 4: The comparison of acceleration effects on OPT models, $N = 5$ and $K = 7$.

A.8 Runtime Update

In Figure 12, we present an experimental comparison to assess the impact of synchronizing updates to the N-gram module (denoted as Runtime Update) during the decoding stage. The comparison involved three distinct models based on LLaMA-7B, evaluated on the CNN/DM dataset. The experimental results reveal that employing a runtime update strategy enhances the acceleration of the inference process. This finding indicates that during inference, the content generated by LLMs can exhibit correlations that provide valuable guidance for the generation of content in subsequent contexts, underscoring the importance of dynamic updates within the decoding process.

A.9 Details for Table 1

In Table 1, our ANPD method utilizes a standardized configuration with $N = 5$ and $K = 7$. For (Zhang et al., 2023a), we have selected $K = 12$, based on the specifications detailed in both the published paper and the open-source code. Additionally, for (Zhang et al., 2023a) the draft model of the LLaMA-2-13b and CodeLLaMA-13B is con-

[5]https://huggingface.co/Manuel030/alpaca-opt-6.7b

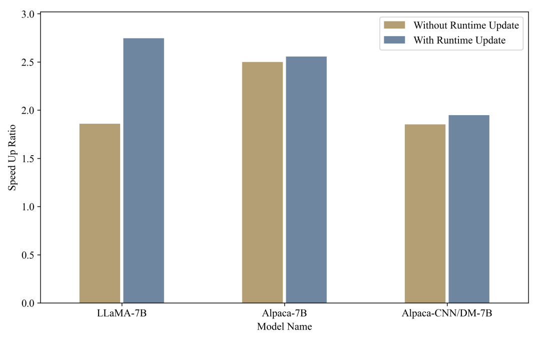


Figure 12: The comparison of acceleration effects for updating the N-gram module during decoding.

structed according to the parameters provided in the open source content[6].

[6]https://github.com/dilab-zju/ self-speculative-decoding