Adaptive Draft-Verification for Efficient Large Language Model Decoding

Xukun Liu[1], **Bowen Lei**[2], **Ruqi Zhang**[3], **Dongkuan (DK) Xu**[4]
[1]Northwestern University,
[2]Texas A&M University
[3]Purdue University,
[4]North Carolina State University
xukunliu2025@u.northwestern.edu, bowenlei@stat.tamu.edu, ruqiz@purdue.edu, dxu27@ncsu.edu

Abstract

Large language model (LLM) decoding involves generating a sequence of tokens based on a given context, where each token is predicted one at a time using the model's learned probabilities. The typical autoregressive decoding method requires a separate forward pass through the model for each token generated, which is computationally inefficient and poses challenges for deploying LLMs in latency-sensitive scenarios. The main limitations of current decoding methods stem from their inefficiencies and resource demands. Existing approaches either necessitate fine-tuning smaller models, which is resource-intensive, or relying on fixed retrieval schemes to construct drafts for the next tokens, which lack adaptability and fail to generalize across different models and contexts. To address these issues, we introduce a novel methodology called `Adaptix`, which accelerates LLM decoding without requiring fine-tuning. Our approach involves an adaptive draft-verification process that evolves over time to improve efficiency. We utilize a tri-gram matrix-based LLM representation to dynamically approximate the output distribution of the LLM, allowing the model to adjust to changing token probabilities during the decoding process. Additionally, we implement a draft construction mechanism that effectively balances exploration and exploitation, ensuring that the drafts generated are both diverse and close to the true output distribution of the LLM. The importance of this design lies in its ability to optimize the draft distribution adaptively, leading to faster and more accurate decoding. Through extensive experiments on various benchmark datasets and LLM architectures, we demonstrate that `Adaptix` accelerates the decoding process while maintaining high accuracy, making it suitable for deployment in a wide range of practical applications.

Code — https://github.com/liuxukun2000/Adaptix

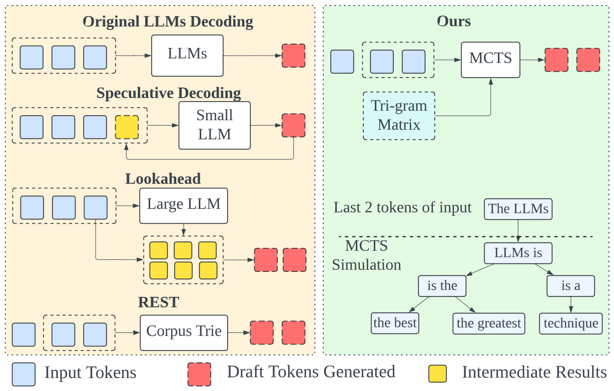


Figure 1: Comparison of different LLM decoding strategies. In *Speculative Decoding*, a small LLM generates predictions (red blocks) from inputs (blue blocks). Yellow blocks indicating intermediate results obtained from language model. *Lookahead* uses a large LLM for forward-looking predictions. *REST* employs a corpus trie for rapid token lookups. `Adaptix` integrates Monte Carlo Tree Search with tri-gram statistics and recent token history to simulate potential outputs, refining its recommendations over time. `Adaptix`'s adaptive approach offers advantages in terms of speed and accuracy by continuously evolving its draft constructions.

Introduction

Large language model (LLM) decoding involves generating a sequence of tokens based on a given context, where each token is predicted one at a time using the model's learned probabilities (Brown et al. 2020; Zhang et al. 2022; Touvron et al. 2023a,b). The core mechanism is autoregressive, where each new token is generated conditioned on the previously generated tokens and the given context. This process is crucial for applications like text generation (Li et al. 2024a; Peng et al. 2023; Chang et al. 2023), machine translation (Zhang, Haddow, and Birch 2023; Moslem et al. 2023; Hendy et al. 2023), and conversational AI (Shanahan 2024; Wu et al. 2023; Saka et al. 2023). However, each decoding step involves a forward pass through the model, making the process inherently sequential and computationally expensive. The inefficiencies arise due to the need to reload the model for each token prediction, leading to high computational costs and memory bandwidth usage. This serial nature of decoding is a significant bottleneck, especially for real-time applications (Liu et al. 2023a; Mandvikar 2023; Antoniol et al. 1994) where latency is critical. Thus, optimizing the decoding speed of LLMs is essential for practical deployment.

Recent research has explored various strategies to mitigate the inefficiencies of LLM decoding. *Speculative De-*

coding (Leviathan, Kalman, and Matias 2023; Spector and Re 2023; Chen et al. 2023) introduces an approach where a smaller, more efficient model generates several token predictions in parallel, which are then verified by the larger target model. This method leverages the efficiency of smaller models to reduce the number of serial forward passes required, achieving substantial speedups without altering the output distribution. *Lookahead Decoding* (Fu et al. 2024a) uses the full context to predict multiple future tokens, creating a buffer that reduces the dependency on sequential processing. *REST* (He et al. 2024) employs a retrieval-based approach where relevant tokens are fetched from a pre-constructed datastore using the current context, forming drafts that are verified by the LLM. These methods can be summarized within the ***draft-verification*** pipeline, as shown in Figure 1. *Speculative Decoding* and *Lookahead* Decoding both generate draft tokens through predictive models, while REST constructs drafts from retrieved tokens based on the context. In each case, the drafts are then verified by the main LLM, ensuring that the final output adheres to the model's learned probabilities. Despite their advancements, these approaches face notable limitations. They often require additional training or fine-tuning, which can be resource-intensive. Fixed retrieval schemes lack adaptability, making it challenging to adjust the draft distribution in real-time based on the evolving LLM output. Additionally, these methods may not generalize well across different models and contexts, limiting their effectiveness in dynamic environments.

In this work, our focus is on ***fine-tuning-free draft-verification*** to address these limitations. The draft-verification pipeline can be viewed as a rejection sampling procedure where the similarity between the proposal distribution (*draft*) and the target distribution (*LLM output*) is crucial for the acceptance rate and convergence speed. Higher similarity results in a higher acceptance rate and faster decoding speed. Very few fine-tuning-free approaches, *e.g.*, *REST* (He et al. 2024), typically use fixed retrieval-based schemes to construct drafts. These schemes lack the adaptability to adjust the draft distribution based on the evolving LLM output distribution, resulting in a persistent gap between the draft and the actual LLM output. This gap reduces the draft acceptance rate and limits the potential for improving decoding speed. To address this issue, we raise the following question:

Research Question: How to design an adaptive draft construction process that can evolve itself and accurately approximate LLM outputs during decoding?

To introduce adaptability and find drafts that are increasingly close to the LLM output distribution during decoding, we not only need to have an adaptive draft construction pipeline but also need to maintain a balance between exploration and exploitation. This balance ensures that speedups can be achieved by leveraging existing knowledge of draft construction while continuously exploring better draft construction capabilities. To achieve this, we propose a novel methodology called `Adaptix`. `Adaptix` incorporates a tri-gram-matrix-based adaptive LLM representative to control the conditional probability distribution of the next token, which can be updated during the decoding process to adjust the draft construction accordingly. To balance exploration and exploitation, we design a *draft maker* inspired by *Monte Carlo Tree Search (`MCTS`)* (Coulom 2007; Browne et al. 2012; James, Konidaris, and Rosman 2017; Świechowski et al. 2023). This *draft maker* uses a token preference score to maintain the balance during the search process. The score consists of two parts: the first part is based on the approximate conditional probability distribution of the next token obtained from the LLM representative, reflecting the *draft maker*'s current knowledge of the LLM output; the second part encourages the *draft maker* to explore unexplored or less-explored draft spaces. Theoretically, we show that our method can be viewed as a constrained optimization problem to encourage the draft distribution to converge to the LLM output distribution. Using the token preference score, the *draft maker* can effectively search the draft space and generate candidate tokens. After the draft construction and verification are completed, the information is fed back to the LLM representative to update its approximation of the LLM output. This feedback loop enriches the *draft maker*'s knowledge in subsequent rounds of draft-verification, enabling adaptability and self-improvement in the draft construction process.

In summary, our contributions are concluded as follows:

- We design a *tri-gram matrix-based* representation that dynamically approximates the LLM output distribution, enhancing adaptability without the need for fine-tuning. It addresses the limitation of fixed retrieval schemes by continuously evolving with the model's predictions.
- We develop a *draft maker* that effectively balances exploration and exploitation to generate high-quality drafts. This mechanism improves decoding speed and accuracy by ensuring that the drafts are closely aligned with the LLM's output distribution. Our experiments show a **2.5X** improvement in decoding speed compared to baselines.
- Through extensive experiments on various benchmark datasets and LLM architectures, we demonstrate that `Adaptix` accelerates the decoding process while maintaining high accuracy. Specifically, we achieve up to a **2.5X** speedup in latency and an average acceptance rate improvement of **20%** over existing methods.
- Our method's ability to adapt to evolving LLM outputs and continuously refine draft construction sets it apart from existing ones, addressing the need for more flexible and dynamic decoding solutions.

Methodology

We propose a new fast fine-tuning-free draft-verification LLM decoding method by introducing adaptability into the decoding and learning from LLM, which is illustrated in Figure 2. Existing accelerated decoding algorithms either require additional fine-tuning or lack adaptability to LLM's output distributions, resulting in additional cost or insufficient acceleration. To address these issues, we design an adaptive LLM representation based on a tri-gram matrix to adaptively approximate the output distribution of the LLM, develop a draft maker that balances exploration and exploita-

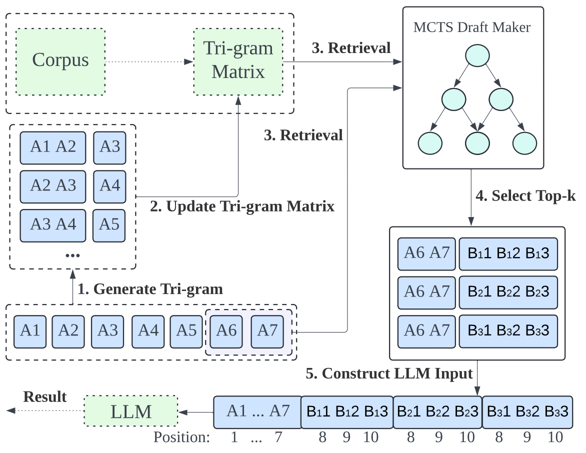


Figure 2: The data processing workflow of Adaptix. Initially, the input tokens undergo preprocessing to calculate their tri-grams, which serve to update the tri-gram matrix. Subsequently, the updated matrix, in conjunction with the last two tokens of the input, is used to retrieve potential token sequences. These sequences are ranked, and the top-k sequences are selected, and then appended to the original input. Finally, these extended sequences are inputted into the Large Language Model for prediction.

tion for self-improvement towards high-quality drafts, and verify the drafts using tree attention.

Preliminary

Speculative decoding is a method to accelerate language model inference by using a smaller auxiliary model to generate a draft sequence, reducing the computational load on the larger model (Leviathan, Kalman, and Matias 2023). ***Retrieval-based speculative decoding*** extends this by incorporating a retrieval system instead of the smaller model, leveraging pre-stored corpus segments for relevant text generation. ***Monte Carlo Tree Search*** (MCTS) (Coulom 2007; Browne et al. 2012; James, Konidaris, and Rosman 2017; Świechowski et al. 2023) is an algorithm that optimizes decision-making by balancing exploration and exploitation of future states. It selects nodes for further exploration using a combination of node visit counts and estimated values, aiming to maximize overall outcomes. For a comprehensive discussion of these methods, please refer to ***Appendix E***.

Adaptive LLM Representative

To approximate the output token distribution of the LLM without fine-tuning the small model, we distill linguistic knowledge from a small corpus and construct a tri-gram matrix as an initial representation of the LLM, which allows us to leverage the statistical regularities of language at a granular level. Specifically, we summarize and count each set of three tokens that appear in the corpus and compute the probability of the third token appearing conditional on the first two tokens. The formula is defined in Eq. (1):

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}, \tag{1}$$

where $P(w_i|w_{i-2}, w_{i-1})$ is the conditional probability of a word w_i given the two preceding words w_{i-2} and w_{i-1}, $C(w_{i-2}, w_{i-1}, w_i)$ is the count of the tri-gram occurrence in the corpus, and $C(w_{i-2}, w_{i-1})$ is the count of the preceding bi-gram (Mori, Nishimura, and Itoh 1998).

In this way, we can obtain a good initial LLM representative at a much lower cost, which can generate an approximate distribution of the next token based on the previous tokens. This LLM representative will collaborate with our draft maker to generate drafts and get feedback to update the tri-gram matrix for adaptability and self-improvement. Please see **Section 2.3** for more details.

Draft Maker and Self-Improvement

With the help of the LLM representative, we further propose a draft maker that balances exploration and exploitation while searching for candidate drafts that are closer to the LLM output. On the one hand, the draft maker leverages the conditional probabilities from the LLM representative, which include current knowledge of the LLM output. On the other hand, the draft maker is encouraged to search more in the unexplored or less explored draft space to find better draft candidates. Then, with the feedback from the LLM output, the LLM representative can update its understanding of the LLM output, improve the draft maker's search, and achieve self-improvement. Details are provided below.

Draft Search Score: Given the initial tokens, we exploit Monte Carlo Tree Search (MCTS) (Coulom 2007) to guide the search process of the drafts of the next tokens, where we prioritize candidate tokens according to the conditional probability from the tri-gram matrix-based LLM representative and the node visitation counts during the tree search. The score plays a key role in balancing exploration and utilization during the Monte Carlo tree search and is defined as Eq. (2).

$$\text{PUCT}(s, a) = Q(s, a) + E \cdot P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}. \tag{2}$$

The score design is motivated by PUCT Score (Rosin 2011; Silver et al. 2017). In particular, $Q(s, a)$ assesses the quality of taking action a in state s, while $P(s, a)$ represents the prior probability of selecting action a in state s. The term $N(s, a)$ denotes the number of times the action a has been taken from state s, and $\sum_b N(s, b)$ sums the counts for all actions from state s. Eq. (3) plays a critical role in determining the balance between exploration and exploitation within the MCTS framework.

$$E = C_1 + \log\left(\frac{\sum_b N(s, b) + C_2 + 1}{C_2}\right), \tag{3}$$

The constant C_1 acts as a base level adjustment, while C_2 modulates the logarithmic term to scale the exploration factor dynamically based on the total visitation counts. This formula ensures that our draft choices are contextually appropriate and optimizes the robustness and coherence of text generation.

Self-Improvement Strategy Transfer: Based on the final search score obtained during the search, we can construct draft candidates and verify them to get the final decoding output (please see **Section 2.4**) and feed it back for self-improvement. This final output decoding represents LLM's output distribution, which would be a good learning material for the LLM representative. Therefore, we feed this knowledge into the LLM representative in order to obtain updated conditional probability distributions, thus providing the draft maker with more accurate and exploitable knowledge, which is illustrated in Figure 2. Specifically, this technique operates by first extracting tri-grams from recent outputs of the LLM. Each tri-gram's frequency is then used to update its probability as potential outputs. These adjusted probabilities are fed into the `MCTS` as part of the policy network, influencing the selection of the tree search. The updated tri-gram probabilities essentially serve as a dynamic policy guide, enhancing the model's ability to generate contextually relevant and coherent sequences. By incorporating learned tri-gram probabilities into the tree search algorithm, we effectively create a feedback loop where the search strategy itself evolves over time. This strategy adjustment is executed by recalibrating the exploration-exploitation balance based on the empirical data derived from the model's own outputs.

Draft Construction and Verification

It is important to note that candidate drafts generated by the draft maker often have common starting segments that can cause redundant recalculations in the Transformer layers if not managed correctly. To address the issue, a pseudo-sequence that guarantees that each draft is a sub-sequence and that any common prefix appears only once is created (He et al. 2024). Motivated by this observation, we use a specific attention mask for each attention layer, called *tree attention* (Miao et al. 2023; Cai et al. 2024). This mask aligns the computations for each token with its dependencies according to the original draft sequence, preserving the draft's contextual integrity and preventing unnecessary computations. The approval of drafts relies on a comparison with the conditional distribution from the LLM. At each position, new tokens are sampled and compared to the draft tokens. If a sampled token corresponds to the draft token, it is approved; otherwise, the draft is discarded from that point. This selective approval ensures that the output sequence aligns with what would be produced by a typical autoregressive process, thus upholding the authenticity of the generated text.

Theoretical Insight: Why `Adaptix` uses `MCTS`

In this section, we provide a theoretical justification for the design of `Adaptix`. We show that the draft search in `Adaptix` using `MCTS` can be viewed as a form of policy optimization, while the inference mechanism of LLMs can be viewed as a similar form of penalty optimization.

`MCTS` in `Adaptix`: The token selection procedure in `Adaptix` decoding can be viewed as an action selection process. The `MCTS` algorithm optimizes its policy by iteratively building a search tree and updating visit counts for each node (state-action pair) based on the search paths. The visit count distribution $\hat{\pi}(a \mid x)$ is defined as:

$$\hat{\pi}(a \mid x) \triangleq \frac{1 + n(x,a)}{|A| + \sum_b n(x,b)}, \tag{4}$$

where $n(x,a)$ represents the visit count for action a in state x, and $|A|$ represents the total number of possible actions at state x. Then, the action selection in `MCTS` can be written as selecting the action a^*:

$$a^*(x) \triangleq \arg\max_a [Q(x,a) + \lambda_N \cdot \frac{\pi_\theta(a \mid x)}{\hat{\pi}(a \mid x)}] \tag{5}$$

Following (Grill et al. 2020), we use $q \in \mathcal{R}^{|A|}$ to denote the vector of Q-function $Q(x,a)$. With proper choice of hyperparameters, the `MCTS` algorithm can be viewed as searching for the optimum solution to a policy optimization problem (Grill et al. 2020) as below:

$$\bar{\pi} \triangleq \arg\max_{y \in S} \left[q^\top y - \lambda_N \mathrm{KL}[\pi_\theta, y] \right], \tag{6}$$

where S is the $|A|$-dimensional simplex, λ_N is a regularization parameter that depends on hyperparameters and balances exploration and exploitation, and KL is the KL-divergence.

LLM Inference Mechanism: Large language models, particularly those based on the Transformer architecture, generate text by predicting the probability distribution of the next token given the previous tokens. During training, the model maximizes the log-likelihood of the observed data, which is equivalent to minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P(w_t \mid w_{1:t-1}; \theta) + \frac{\lambda}{2} \|\theta\|_2^2, \tag{7}$$

where P denotes the conditional probability of LLM, w denotes the tokens, and θ denotes the model parameters.

Comparative Analysis: As shown in Eq (6) and Eq. (7), both `MCTS` and LLMs can be viewed as regularized optimization problems for selecting the distribution of the next tokens. On the one hand, the Q-function in `MCTS` for `Adaptix` can be viewed as an approximation to the log-likelihood of LLMs:

$$\begin{aligned} Q(x,a) &= -\sum_{t=2}^{T} \log \hat{P}(w_t \mid w_{t-1}, w_{t-2}; \theta) \\ &\approx \log P(w_0, w_1, \cdots, w_T; \theta) \\ &= -\sum_{t=2}^{T} \log P(w_t \mid w_{1:t-1}; \theta), \end{aligned} \tag{8}$$

where $\hat{P}$ and P are the conditional probability distribution from tri-gram-matrix-based LLM representative and LLMs, respectively. On the other hand, both `MCTS` and LLMs employ regularization to improve the optimization procedure. As a result, we verify the similarities between `MCTS` and LLM Inference in terms of optimization and regularization.

Experiments

Experimental Setup

Models and Datasets. We conduct a series of experiments with five distinct models on three datasets to evaluate the efficacy of `Adaptix`. In particular, We use three Vicuna models (Chiang et al. 2023) (7B, 13B, 33B) and two LLaMA2-chat models (Touvron et al. 2023b) (7B, 13B) to evaluate the acceleration capabilities across different model sizes and types. Our assessment incorporates the HumanEval (Chen et al. 2021), MT-Bench (Zheng et al. 2023),

		Latency					Average Accept Length			
Benchmark	Model	REST	REST Single	Lookahead	Autoregressive	`Adaptix`	REST	REST Single	Lookahead	`Adaptix`
MT-Bench	Vicuna-7B	16.31	17.36	18.93	24.77	**12.95**	1.97	1.98	1.89	**2.42**
	Vicuna-13B	25.43	25.99	32.73	44.07	**22.94**	1.98	1.99	1.85	**2.39**
	Vicuna-33B	28.63	28.62	40.53	52.97	**24.96**	1.95	1.96	1.83	**2.29**
	Llama2-7B	16.08	17.67	18.84	25.58	**13.85**	1.96	1.95	1.96	**2.30**
	Llama2-13B	27.13	29.80	31.24	44.76	**25.13**	1.95	1.95	1.96	**2.32**
Alpaca	Vicuna-7B	14.24	14.58	18.73	24.49	**12.81**	2.22	2.22	1.89	**2.33**
	Vicuna-13B	**22.94**	23.01	32.60	43.60	24.06	2.21	2.21	1.86	**2.26**
	Vicuna-33B	26.03	25.89	40.58	52.52	**24.62**	2.11	2.12	1.82	**2.21**
	Llama2-7B	14.13	14.87	19.28	25.38	**12.90**	2.21	2.20	1.97	**2.37**
	Llama2-13B	23.66	24.07	31.18	44.04	**23.57**	2.15	2.13	1.96	**2.32**
Human Eval	Vicuna-7B	14.90	15.56	18.99	25.49	**11.24**	2.21	2.23	2.10	**2.67**
	Vicuna-13B	20.17	20.61	27.43	45.13	**19.96**	2.50	2.50	2.23	**2.81**
	Vicuna-33B	24.91	25.06	31.34	52.32	**21.19**	2.29	2.30	2.02	**2.62**
	Llama2-7B	14.37	15.57	15.28	25.91	**11.68**	2.19	2.19	2.27	**2.63**
	Llama2-13B	25.46	25.85	26.72	45.25	**21.82**	2.01	2.01	2.17	**2.60**

Table 1: Latency and Average Accept Length Comparison between `Adaptix` and Baselines. In most test cases, `Adaptix` has the lowest latency, longer accept length, and higher efficiency.

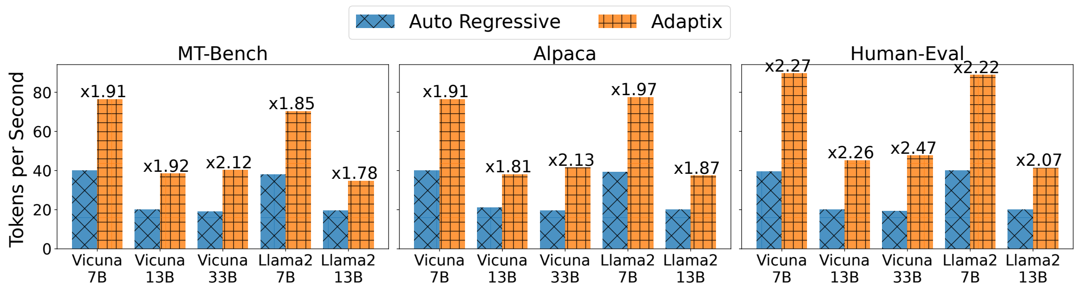


Figure 3: Comparison of `Adaptix`'s throughput for different models on MT-Bench, Alpaca, and Human-Eval. The performance of `Adaptix` shows stable and significant improvements across different models and benchmarks.

and Alpaca (Taori et al. 2023) datasets to ascertain general natural language understanding and generation competencies. These datasets are meticulously chosen to guarantee a comprehensive analysis of the acceleration techniques across various tasks.

Corpus. We construct two corpora. The first one is built using a portion of the Python pre-training code from The Stack (Kocetkov et al. 2022), comprising about 2.7M Python code samples with a resulting size of 1007MB. The second is constructed using data derived from UltraChat (Ding et al. 2023), consisting of around 774K ChatGPT conversations, producing a corpus with a size of 574MB. The experiments on the MT-Bench and Alpaca are conducted using the UltraChat corpus, while the Human-Eval benchmark utilize the corpus from The Stack.

Metrics. To assess the acceleration performance on large language models, we use two main metrics: speedup ratio and average acceptance length. Speedup ratio, calculated as the ratio of the time required by the baseline models to complete inference tasks without acceleration to the time required by our `Adaptix`, measures the efficiency gains introduced by the algorithm. The second metric, average acceptance length, measures the average number of tokens accepted per forward pass by the target large language models, excluding any overhead of retrieving and constructing draft tokens, indicating the maximum possible acceleration.

Baselines. We compare various foundational approaches to improve the decoding speed of large language models. We examine *Lookahead Decoding* (Fu et al. 2024a), a precise and parallel decoding algorithm that cuts down latency without relying on draft models. We compare *REST* (He et al. 2024) (Retrieval-Based Speculative Decoding), which adopts a retrieval-based strategy to create draft tokens, in contrast to conventional speculative decoding methods that rely on a draft model.For fairness in comparison, we include *REST Single*, a single-threaded version of REST, to evaluate performance under constrained processing conditions. We also include the traditional *Autoregressive* method, which represents the standard decoding approach, serving as a baseline to highlight the improvements offered by the other methods. All experiments are conducted on NVIDIA A6000 GPUs, except for the 33B model, which utilizes an NVIDIA H100. The experiments default to Greedy sampling.

Main Results[1]

In the experiments, we compare the efficacy of different baselines applied to various models, utilizing three datasets: MT-Bench, Human-Eval, and Alpaca. We focus on metrics of Accept Length, Latency, and Speedup Ratio. Table 1

[1]Due to space constraints, we provide detailed settings for each set of experiments in ***Appendix F***.

summarizes the latency and average accept length on the three datasets. Adaptix consistently demonstrates lower latency, particularly for the vicuna-7B and llama2-13B models. For instance, on MT-Bench, Adaptix achieves a latency of 12.95 ms for vicuna-7B, which is lower than *REST* (16.31 ms), *REST* Single Thread (17.36 ms), and *Lookahead* (18.93 ms). Notably, **the memory required for Adaptix (574MB) is only 5.6% of that required for REST (12GB). According to Table 2, even when Adaptix uses a smaller corpus (253MB, which is just 2.5% of REST's requirements), it still achieves lower latency than REST**. This trend is also observed on Alpaca, where Adaptix achieves a latency of 12.81 ms for vicuna-7B, compared to 14.24 ms for REST, 14.58 ms for *REST* Single Thread, and 18.73 ms for *Lookahead*.

The accept length results in Table 1 indicate the quality of the generated outputs, with longer accept lengths suggesting more coherent and contextually relevant text. Our method, Adaptix, outperforms other methods across different models on both MT-Bench and Alpaca datasets. For example, on MT-Bench, Adaptix achieves the highest accept length for vicuna-33B and llama2-13B models, showcasing its superior language generation capabilities.

Speedup ratio are used to evaluate the efficiency. Adaptix consistently shows a significant improvement in speed up across all datasets in Figure 3. This efficiency is noticeable on the MT-Bench, Alpaca and Human-Eval datasets, where Adaptix not only reduces latency but also enhances the overall processing speed. For instance, Adaptix achieves a speedup of 1.92x on MT-Bench with the vicuna-13B model, outperforming *REST*, *REST* Single Thread, and *Lookahead*. On the HumanEval dataset, the vicuna-33B model, for example, demonstrates a speedup of nearly 2.5x when using Adaptix.

Stability of Adaptix

In this section, we analyze the stability of In this section, we evaluate the stability of Adaptix across various task categories, including writing, roleplay, reasoning, math, coding, extraction, STEM, and humanities. Adaptix maintains consistent performance across all categories, achieving the highest speedup in coding (×2.69) and the lowest in extraction (×2.17). The average accept length remains stable, confirming that Adaptix can effectively handle diverse tasks without significant performance variations.

To further evaluate the robustness of Adaptix, we analyze the effects of top-p and temperature on performance. Figures 4b and 4c show that variations in these parameters have minimal impact on the average accept length. Specifically, Figure 4b indicates that Adaptix's performance remains stable across different top-p values, while Figure 4c demonstrates similar consistency for temperature changes. These results confirm the robustness of Adaptix to parameter variations.

Ablation Study

To gain insight into our method, we conduct a series of ablation studies. Full studies are summarized in ***Appendix D***.

Effect of the adaptive strategy. Figure 4a illustrates the performance of our adaptive strategy on Vicuna-7B, with the analysis of average accept lengths over varying token counts. We find that the adaptive strategy maintains a higher average accept length over the entire range compared to the non-adaptive strategy. The adaptive strategy's success is attributed to its dynamic adjustment of the model's probability distributions based on the tri-gram frequencies from prior outputs. This allows the model to better manage longer contexts and maintain relevance, enhancing stability and coherence in longer interactions.

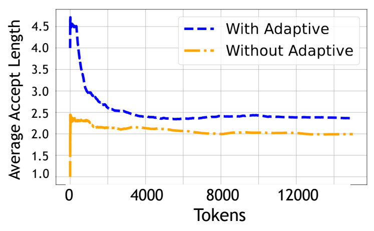
Effect of the corpus size. Table 2 shows the impact of the increase in corpus size from 121k to 774k on various performance metrics. With the expansion of the corpus, there is a gradual improvement in the Accept Length from 2.30 to 2.42. This increase suggests that larger datasets provide a broader array of language patterns, which enhances the model's ability to generate more coherent and contextually relevant outputs. Despite the increase in data size from 253 MB to 574 MB, the system maintains efficient data processing capability. The small differences in latency affirm Adaptix's consistent performance, even with smaller corpus sizes, which further extends its potential for use on resource-constrained devices. The modest rise in retrieval time underscores the efficiency of the retrieval algorithms, which can manage larger datasets without significantly compromising response speed. In summary, the results show that larger corpus sizes can improve the quality of the model's output while maintaining good system performance.

Corpus Size	Corpus Size	Latency	Accept Length	Speed up
121k	253 MB	13.09 ms	2.30	1.89
774k	574 MB	12.95 ms	2.42	1.93

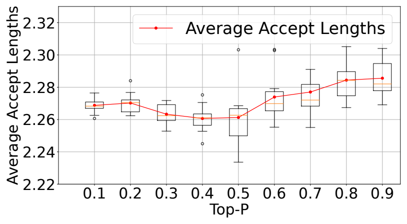
Table 2: Effect of Corpus Size.

Effect of MCTS. Figure 5 presents the results for Vicuna-7B model on the MT-Bench dataset, showing the impact of different MCTS search counts on performance. Increasing the number of searches improves performance, while the optimal number varies by model size. The average accept length and latency are plotted against the number of searches, illustrating the trade-off between performance and computational cost. We further compare greedy search with MCTS (full traversal is not feasible due to the huge search space) while keeping the number of 150 search iterations constant. The results show that the average accept length of the greedy search is only 1.493, significantly lower than that obtained by MCTS, demonstrating the superiority of MCTS in efficiently managing the vast decision spaces.

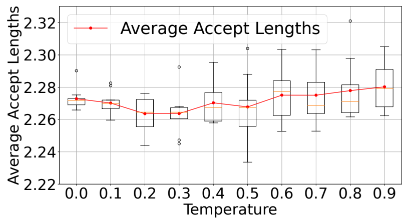
Effect of N-gram Model Choice. Our studies extensively evaluate the impact of different n-gram configurations on decoding performance. In tests conducted on the MT-Bench dataset using the Vicuna-7B model, bi-grams and 4-grams result in accept lengths of 1.80 and 1.82, respectively. These results are significantly lower compared to the 2.30 accept length achieved with tri-grams. Bi-gram models demonstrate limited capability in effectively utilizing contextual information, often leading to outputs that appear more random



(a) Effect of the adaptive strategy

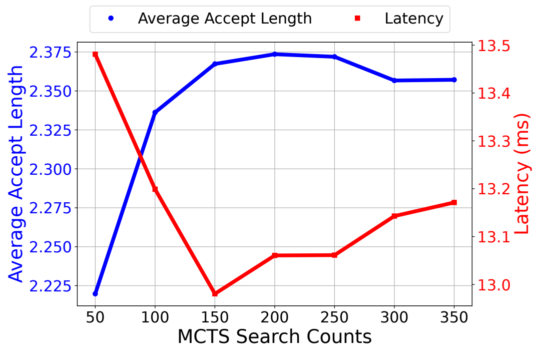


(b) Top-p



(c) Temperature

Figure 4: (4a) Adaptive Strategy comparison on MTBench: Performance of Vicuna-7B model with and without the adaptive strategy on the MT-Bench dataset, showing the advantage of using the adaptive approach. (4b) (4c) Sensitivity analysis of `Adaptix` on top-p and temperature.



(a) Vicuna-7B

Figure 5: Comparison of Search Counts on MT-Bench.

and less coherent. Conversely, 4-grams exhibit overly deterministic behavior, constraining the diversity of the generated text due to their restrictive nature in capturing extensive prior context. Tri-grams strike an optimal balance, providing enough contextual depth to enhance the coherence and relevance of outputs while still allowing for sufficient variability and diversity in text generation. This balance makes tri-grams particularly effective in large language model decoding, as they encapsulate the ideal compromise between randomness and contextual awareness.

Related Work

A number of research efforts on decoding strategies for large language models have used draft models to improve decoding efficiency. Techniques such as Speculative Decoding (Leviathan, Kalman, and Matias 2023; Spector and Re 2023; Chen et al. 2023; Stern, Shazeer, and Uszkoreit 2018), Madusa (Cai et al. 2024), Eagle (Li et al. 2024b), various other approaches requiring draft models (Zhang et al. 2024; Liu et al. 2023b; Kim et al. 2024; Fu et al. 2024b) fall into this category, utilizing models to generate drafts. Specifically, *Speculative Decoding* uses an advanced sampling technique where the auxiliary model generates a set of potential token sequences, and the primary model selects the most sequences, resulting in a good balance between speed and accuracy. Although these methods primarily aim to enhance the accuracy of generated texts and significantly accelerate the response time during initial text generation, their adoption comes with drawbacks. The primary issue is the necessity for additional training specific to the draft models, which could be resource-intensive. Moreover, these techniques generally depend on GPU resources (Kwon et al. 2023; Sheng et al. 2023; Park et al. 2024) for inference, potentially limiting their application in environments where such hardware is unavailable or when operating under strict resource constraints.

A significant portion of recent advances has focused on improving efficiency without relying on draft models (Fu et al. 2024a; He et al. 2024). Two notable approaches in this realm are *Lookahead decoding* (Fu et al. 2024a) and *Retrieval-Based Speculative Decoding* (REST) (He et al. 2024). *Lookahead decoding* is an approach that enhances the efficiency of the decoding process through the prediction of subsequent tokens via Jacobi Iteration (Sleijpen and Van der Vorst 2000). It employs a heuristic to estimate the future cost of a sequence without the need to explicitly create a draft. *REST* introduces a retrieval-enhanced generation model that speculatively decodes sequences without the need for producing preliminary drafts. It instead searches and prioritizes possible continuations from an already established sequence database. However, these methods exhibit lower accuracy and greater resource use compared to our approach. They demand more memory and GPU processing power, posing challenges in resource-scarce settings.

Conclusion

`Adaptix` improves the LLM decoding process by introducing adaptability and efficiency, significantly reducing latency and computational demands. This method achieves up to a 2.5X speedup in decoding and a 20% improvement in acceptance rates, outperforming traditional techniques. Unlike existing approaches, `Adaptix` dynamically adjusts the draft distribution using a tri-gram matrix and enhances draft quality through `MCTS`, eliminating the need for fine-tuning. The continuous feedback loop ensures ongoing improvements in draft generation. While `Adaptix` demonstrates robust performance across various benchmarks, future work will focus on exploring its application in more diverse real-world scenarios. Addressing potential limitations in extremely large-scale deployments will be a priority.

Acknowledgments

This work was partially supported by the National Science Foundation (NSF) grants BCS-2416846 and OAC-2417850. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the government.

References

Antoniol, G.; Brugnara, F.; Cettolo, M.; and Federico, M. 1994. Language model estimations and representations for real-time continuous speech recognition. In *ICSLP*, 859–862.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Cai, T.; Li, Y.; Geng, Z.; Peng, H.; Lee, J. D.; Chen, D.; and Dao, T. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. *arXiv preprint arXiv: 2401.10774*.

Chang, J. D.; Brantley, K.; Ramamurthy, R.; Misra, D.; and Sun, W. 2023. Learning to generate better than your llm. *arXiv preprint arXiv:2306.11816*.

Chen, C.; Borgeaud, S.; Irving, G.; Lespiau, J.-B.; Sifre, L.; and Jumper, J. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. arXiv:2302.01318.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code.

Chiang, W.-L.; Li, Z.; Lin, Z.; Sheng, Y.; Wu, Z.; Zhang, H.; Zheng, L.; Zhuang, S.; Zhuang, Y.; Gonzalez, J. E.; Stoica, I.; and Xing, E. P. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality.

Coulom, R. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In van den Herik, H. J.; Ciancarini, P.; and Donkers, H. H. L. M. J., eds., *Computers and Games*, 72–83. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

Ding, N.; Chen, Y.; Xu, B.; Qin, Y.; Zheng, Z.; Hu, S.; Liu, Z.; Sun, M.; and Zhou, B. 2023. Enhancing Chat Language Models by Scaling High-quality Instructional Conversations. *arXiv preprint arXiv:2305.14233*.

Fu, Y.; Bailis, P.; Stoica, I.; and Zhang, H. 2024a. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. arXiv:2402.02057.

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

Grill, J.; Altché, F.; Tang, Y.; Hubert, T.; Valko, M.; Antonoglou, I.; and Munos, R. 2020. Monte-Carlo Tree Search as Regularized Policy Optimization. *CoRR*, abs/2007.12509.

He, Z.; Zhong, Z.; Cai, T.; Lee, J.; and He, D. 2024. REST: Retrieval-Based Speculative Decoding. In Duh, K.; Gomez, H.; and Bethard, S., eds., *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 1582–1595. Mexico City, Mexico: Association for Computational Linguistics.

Hendy, A.; Abdelrehim, M.; Sharaf, A.; Raunak, V.; Gabr, M.; Matsushita, H.; Kim, Y. J.; Afify, M.; and Awadalla, H. H. 2023. How Good Are GPT Models at Machine Translation? A Comprehensive Evaluation. arXiv:2302.09210.

James, S.; Konidaris, G.; and Rosman, B. 2017. An analysis of monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Kim, S.; Mangalam, K.; Moon, S.; Malik, J.; Mahoney, M. W.; Gholami, A.; and Keutzer, K. 2024. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36.

Kocetkov, D.; Li, R.; Allal, L. B.; Li, J.; Mou, C.; Ferrandis, C. M.; Jernite, Y.; Mitchell, M.; Hughes, S.; Wolf, T.; Bahdanau, D.; von Werra, L.; and de Vries, H. 2022. The Stack: 3 TB of permissively licensed source code. arXiv:2211.15533.

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

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023. Fast Inference from Transformers via Speculative Decoding. arXiv:2211.17192.

Li, J.; Tang, T.; Zhao, W. X.; Nie, J.-Y.; and Wen, J.-R. 2024a. Pre-trained Language Models for Text Generation: A Survey. *ACM Computing Surveys*, 56(9): 1–39.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024b. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. *arXiv preprint arXiv:2401.15077*.

Liu, J.; Yu, C.; Gao, J.; Xie, Y.; Liao, Q.; Wu, Y.; and Wang, Y. 2023a. Llm-powered hierarchical language agent for real-time human-ai coordination. *arXiv preprint arXiv:2312.15224*.

Liu, X.; Hu, L.; Bailis, P.; Stoica, I.; Deng, Z.; Cheung, A.; and Zhang, H. 2023b. Online Speculative Decoding. arXiv:2310.07177.

Mandvikar, S. 2023. Factors to Consider When Selecting a Large Language Model: A Comparative Analysis. *International Journal of Intelligent Automation and Computing*, 6(3): 37–40.

Miao, X.; Oliaro, G.; Zhang, Z.; Cheng, X.; Wang, Z.; Wong, R. Y. Y.; Chen, Z.; Arfeen, D.; Abhyankar, R.; and Jia, Z. 2023. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 1(2): 4.

Mori, S.; Nishimura, M.; and Itoh, N. 1998. Word clustering for a word bi-gram model. In *ICSLP*. Citeseer.

Moslem, Y.; Haque, R.; Kelleher, J. D.; and Way, A. 2023. Adaptive Machine Translation with Large Language Models. arXiv:2301.13294.

Park, Y.; Hyun, J.; Cho, S.; Sim, B.; and Lee, J. W. 2024. Any-Precision LLM: Low-Cost Deployment of Multiple, Different-Sized LLMs. *arXiv preprint arXiv:2402.10517*.

Peng, C.; Yang, X.; Chen, A.; Smith, K. E.; PourNejatian, N.; Costa, A. B.; Martin, C.; Flores, M. G.; Zhang, Y.; Magoc, T.; et al. 2023. A study of generative large language model for medical research and healthcare. *NPJ Digital Medicine*, 6(1): 210.

Rosin, C. D. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61: 203–230.

Saka, A. B.; Oyedele, L. O.; Akanbi, L. A.; Ganiyu, S. A.; Chan, D. W.; and Bello, S. A. 2023. Conversational artificial intelligence in the AEC industry: A review of present status, challenges and opportunities. *Advanced Engineering Informatics*, 55: 101869.

Shanahan, M. 2024. Talking about large language models. *Communications of the ACM*, 67(2): 68–79.

Sheng, Y.; Zheng, L.; Yuan, B.; Li, Z.; Ryabinin, M.; Chen, B.; Liang, P.; Ré, C.; Stoica, I.; and Zhang, C. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, 31094–31116. PMLR.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Sleijpen, G. L.; and Van der Vorst, H. A. 2000. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM review*, 42(2): 267–293.

Spector, B.; and Re, C. 2023. Accelerating LLM Inference with Staged Speculative Decoding. arXiv:2308.04623.

Stern, M.; Shazeer, N.; and Uszkoreit, J. 2018. Blockwise Parallel Decoding for Deep Autoregressive Models. arXiv:1811.03115.

Świechowski, M.; Godlewski, K.; Sawicki, B.; and Mańdziuk, J. 2023. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3): 2497–2562.

Taori, R.; Gulrajani, I.; Zhang, T.; Dubois, Y.; Li, X.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023a. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

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

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; Awadallah, A. H.; White, R. W.; Burger, D.; and Wang, C. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

Zhang, A.; Wang, C.; Wang, Y.; Zhang, X.; and Cheng, Y. 2024. Recurrent drafter for fast speculative decoding in large language models. *arXiv preprint arXiv:2403.09919*.

Zhang, B.; Haddow, B.; and Birch, A. 2023. Prompting large language model for machine translation: A case study. In *International Conference on Machine Learning*, 41092–41110. PMLR.

Zhang, S.; Roller, S.; Goyal, N.; Artetxe, M.; Chen, M.; Chen, S.; Dewan, C.; Diab, M.; Li, X.; Lin, X. V.; Mihaylov, T.; Ott, M.; Shleifer, S.; Shuster, K.; Simig, D.; Koura, P. S.; Sridhar, A.; Wang, T.; and Zettlemoyer, L. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E. P.; Zhang, H.; Gonzalez, J. E.; and Stoica, I. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685.