ADASPEC: Adaptive Multilingual Speculative Decoding with Self-Synthesized Language-Aware Training and Vocabulary Simplification

Dinh-Truong Do*, Nguyen-Khang Le*, Le-Minh Nguyen
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
truongdo@jaist.ac.jp, lnkhang@jaist.ac.jp, nguyenml@jaist.ac.jp

Abstract

Speculative decoding accelerates large language model (LLM) inference by using a lightweight drafter to propose multiple tokens, which are then verified in parallel by the base model. While effective in English, existing methods often struggle in multilingual scenarios due to static vocabularies and the lack of language-specific instruction data. To address these limitations, we present **ADASPEC**, a multilingual speculative decoding framework that dynamically adapts both the drafter and vocabulary at decoding time. ADASPEC generates language-specific instruction data using the LLM itself, enabling training of drafters for low-resource languages. It also constructs adaptive vocabularies tailored to each language's characteristics. In addition, we introduce MULTI-SPECBENCH, a comprehensive multilingual benchmark covering seven languages and seven generation tasks, to evaluate multilingual speculative decoding performance. Extensive experiments show that ADASPEC achieves up to 2.3× speedup over the state-of-the-art method of EAGLE-2, even in English, demonstrating its effectiveness across diverse languages and tasks.

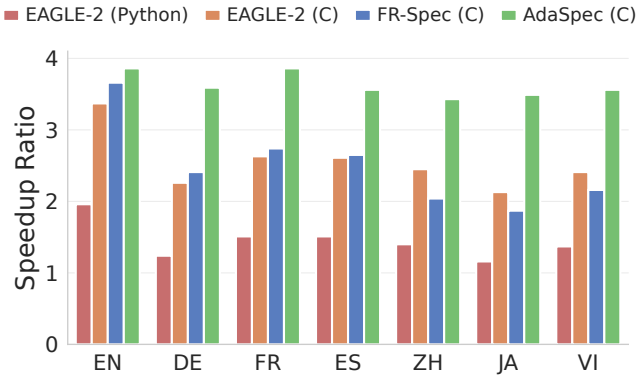


Figure 1: Speedup ratios across languages in monolingual settings (LLaMa-3-8B). While EAGLE-2 and FR-Spec degrade in non-English cases, ADASPEC (C/CUDA) achieves consistently high performance, showing the benefit of multilingual adaptation and low-level optimization.

1 Introduction

Large language models (LLMs) have made significant advances in artificial intelligence, enabling systems to perform a wide range of tasks such as code generation, math reasoning, and open-domain question answering (Yang et al. 2025a; Guo et al. 2025; Touvron et al. 2023; Achiam et al. 2023). These models have evolved from monolingual capabilities to supporting multiple languages to meet the needs of real-world users. With increased ability and multilingual capability comes increased model size and inference latency. To address this, speculative decoding (Chen et al. 2023a; Leviathan, Kalman, and Matias 2023) has emerged as a promising technique to speed up inference, adopting a guess-and-verify strategy. In this approach, a lightweight drafter model first proposes several draft sequences, which the target LLM then verifies in parallel. The longest subsequence that aligns with the target LLM's own predictions is accepted. While speculative decoding has attracted considerable recent interest, its adaptation to multilingual scenarios—common in practical applications—remains underexplored. While state-of-the-art methods like EAGLE-2 (Li et al. 2024d,b) and FR-SPEC (Zhao et al. 2025) have proven effective for English, their performance in multilingual settings remains limited. These methods rely on training drafters using instruction datasets like ShareGPT, which are primarily English-focused. As shown in Figure 1, the performance of these methods drops significantly in non-English languages, revealing the inadequacy of these training methods.

To overcome these limitations, we introduce **ADASPEC**, a framework for speculative decoding that is explicitly designed for multilingual scenarios. ADASPEC dynamically adapts both the drafter model and the decoding vocabulary at each generation step. To create language-specific drafters, we propose a technique to use the target LLM itself to generate instruction-tuning data in any desired language, including low-resource ones. With these synthetic datasets, ADASPEC trains drafters specialized in each language. We also address the inefficiency of fixed vocabularies in multilingual contexts. Prior works have shown that restricting the output vocabulary of the drafter improves speed (Zhao et al. 2025), but these methods rely on a static subset of tokens and a fixed vocabulary size. In contrast, ADASPEC con-

*These authors contributed equally.

structs adaptive vocabularies tailored to the target language and dynamically selects both the vocabulary and its size during decoding based on the generation context. Moreover, to reduce the overhead caused by Python's interpreter (Zhao et al. 2025), we optimized our framework in C/CUDA with preallocated memory. This low-level optimization can significantly reduce runtime latency compared to Python code (Figure 1).

To support robust evaluation, we introduce **MULTI-SPECBENCH**, a multilingual extension of the SpecBench benchmark (Xia et al. 2024). It includes seven languages (English, German, French, Spanish, Chinese, Japanese, and Vietnamese) and spans diverse real-world tasks such as question answering, code generation, summarization, and math reasoning. Our experiments evaluate AdaSpec across two major LLM families—LLaMA-3 (Dubey et al. 2024) and Qwen-2.5 (Yang et al. 2025b)—ranging from 1B to 14B parameters. Results show that ADASPEC consistently achieves the highest speedups across all languages and tasks, outperforming strong baselines like EAGLE-2 and FR-SPEC. ADASPEC delivers up to **2.3×** speedup over EAGLE-2, with robust performance on multiple tasks. In summary, the key contributions of this paper are as follows:

- We propose **ADASPEC**, the multilingual speculative decoding framework with (1) language-specific drafters trained with self-synthesized data, and (2) Language-aware vocabulary simplification in LM head.
- We introduce an adaptive mechanism to select the optimal drafter and the vocabulary set, during the generation process, based on the current context.
- We implement an optimized C/CUDA backend, reducing latency over Python implementations, and efficient on-the-fly drafter changing and LM Head simplification.
- Experiments on seven languages show that ADASPEC achieves superior speedups over existing methods.

2 Preliminaries

Speculative Decoding with Drafter. Speculative Decoding is a class of techniques designed to accelerate autoregressive generation by leveraging an auxiliary model trained to approximate the behavior of a larger language model (LLM). Let $\mathcal{M}$ denote a pretrained LLM that we want to speed up, referred to as the target model. The model has vocabulary $\mathcal{V}$ and consists of an embedding layer $\mathcal{E}$, a stack of L transformer layers $\mathcal{M}^{(1)}_{\text{layer}}, \ldots, \mathcal{M}^{(L)}_{\text{layer}}$, and a language modeling head with a weight matrix $\mathbf{W}_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}| \times d}$. For an input token sequence $\mathbf{x} \in \mathbb{R}^n$, the model computes hidden states $\mathbf{H}_{\mathcal{M}}(\mathbf{x}) \in \mathbb{R}^{n \times d}$ as follows:

$$\mathbf{H}_{\mathcal{M}}(\mathbf{x}) = \mathcal{M}^{(L)}_{\text{layer}} \circ \cdots \circ \mathcal{M}^{(1)}_{\text{layer}}(\mathcal{E}(\mathbf{x})), \quad (1)$$

$$\mathcal{M}(\mathbf{x}) = \text{Softmax}(\mathbf{H}_{\mathcal{M}}(\mathbf{x})\mathbf{W}^{\top}_{\text{LM}}). \quad (2)$$

To approximate $\mathcal{M}$ with reduced computational cost, speculative decoding methods introduce a lightweight draft model $\mathcal{D}$, typically implemented as a shallow transformer (e.g., single-layer) with the same hidden dimension d. The embedding layer and LM head of $\mathcal{D}$ are often tied to $\mathcal{M}$ and kept frozen. Only the transformer block in $\mathcal{D}$ is trained to mimic the behavior of $\mathcal{M}$ on target training data. The draft model computes the output distribution $\mathcal{D}(\mathbf{x})$ as follows:

$$\mathbf{H}_{\mathcal{D}}(\mathbf{x}) = \mathcal{D}^{(1)}_{\text{layer}}(\mathcal{E}(\mathbf{x})), \quad (3)$$

$$\mathcal{D}(\mathbf{x}) = \text{Softmax}(\mathbf{H}_{\mathcal{D}}(\mathbf{x})\mathbf{W}^{\top}_{\text{LM}}). \quad (4)$$

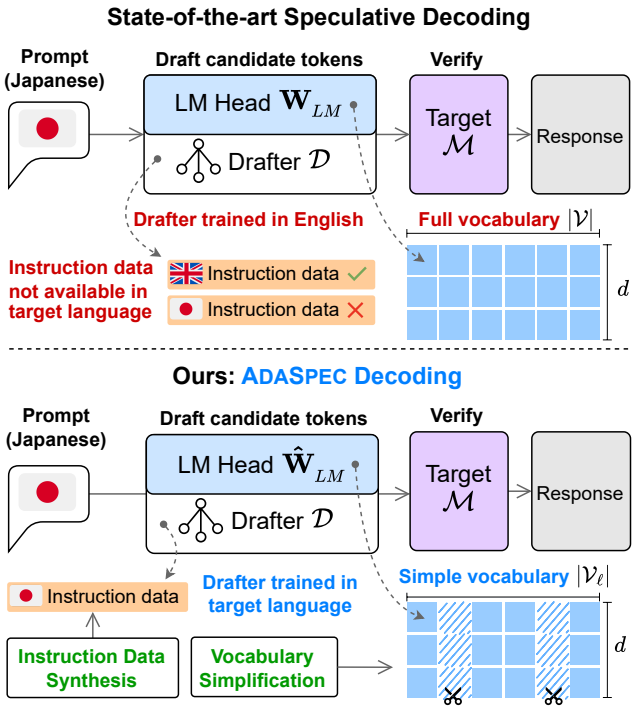


Figure 2: Overview of ADASPEC (bottom), compared to State-of-the-art Speculative Decoding approaches (top).

Modern speculative decoding frameworks (Miao et al. 2024; Li et al. 2024d,b, 2025; Zhang et al. 2025) extend this paradigm by employing the drafter $\mathcal{D}$ multiple times to propose a tree of top-k candidate tokens at each step, forming a draft tree. The target model $\mathcal{M}$ then verifies this draft using tree attention mechanisms (Miao et al. 2024). This enables the model to accept multiple tokens in a single forward pass, significantly boosting inference speed.

The drafter $\mathcal{D}$ is typically trained on instruction data, such as ShareGPT (Li et al. 2024d,b, 2025; Zhang et al. 2025). While high-quality instruction datasets are widely available in English, such resources are often limited or unavailable in other languages—especially in low-resource settings. Our study shows that drafter performance varies significantly across languages and is strongly dependent on the quality and availability of training data, highlighting a major challenge for multilingual speculative decoding.

Vocabulary Simplification. A recent approach (Zhao et al. 2025) aims to reduce the computational burden during speculative decoding by limiting the token set used in the draft model's output layer. Let $\mathcal{V}$ be the full vocabulary of the language model, and $\mathcal{V}_{\text{reduce}} \subset \mathcal{V}$ be a subset

of important tokens. During generation, instead of computing probabilities over $\mathcal{V}$, the draft model restricts its output distribution $\mathcal{D}(\mathbf{x})$ to $\mathcal{V}_{\text{reduce}}$, while the full vocabulary is retained for the verification model. This is done by creating a projection matrix $\hat{\mathbf{W}}_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}_{\text{reduce}}| \times d}$ from $\mathbf{W}_{\text{LM}}$ by only taking the row corresponding to $\mathcal{V}_{\text{reduce}}$. Current approaches are only designed to work with the English language and use a fixed size for $\mathcal{V}_{\text{reduce}}$. The important vocabulary for each language is significantly different from the other. We also find that the number of important tokens ($|\mathcal{V}_{\text{reduce}}|$) required to cover the majority of a specific language is also different from language to language, making it challenging to adapt the approach specific language.

3 AdaSpec Decoding

We propose an approach for efficient speculative decoding in a specific language. We aim to solve two main challenges: (1) training a drafter specifically for a language, which requires instruction data in that language, which is not publicly available for every language; and (2) adapting the LM head to the vocabulary of the specific language.

Drafter for a Specific Language

We find that instruction-tuning a drafter on a specific language significantly improves token acceptance rates, leading to substantial speedup (Figure 3). However, such training requires language-specific instruction data, which is often unavailable—especially for low-resource languages where public datasets are scarce. To address this, we propose generating instruction data from scratch using the target LLM $\mathcal{M}$. As shown in Algorithm 1, our approach begins by translating a base English system prompt and few-shot examples into the target language (lines 2–10). The target LLM then iteratively synthesizes instruction-response pairs in the target language (lines 11–20), with each pair verified by a probabilistic language detector (lines 14–19). Only verified pairs matching the target language are added to the dataset, continuing until we reach the target size. We include detailed illustrations and further analysis of the data synthesis pipeline in the Appendix.

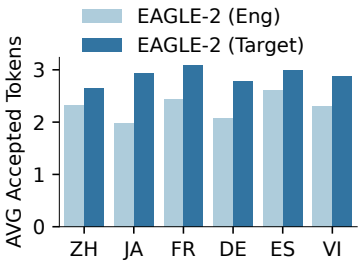


Figure 3: Mean accepted tokens using EAGLE-2 on Llama3 with drafter trained in English (Eng) and drafter trained on target language (Target).

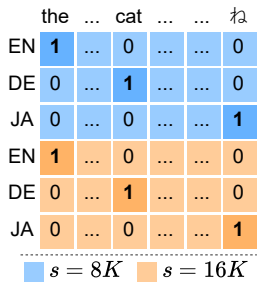


Figure 4: Example of language mask for 3 languages (EN, DE, JA) and 2 vocabulary sizes (8K and 16K).

Algorithm 1: Generate Instruction Data in Target Language

Require: Original model $\mathcal{M}$; Target language ℓ
Require: System prompt (English) $\mathcal{P}_{\text{sys}}^{\text{EN}}$; Translation prompt $\mathcal{P}_{\text{trans}}$
Require: Few-shot examples in English $\mathbf{E}^{\text{EN}} = \{(I_i, R_i)\}_{i=1}^{k}$
Require: Language detector $\mathcal{F}_{\text{detect}}$; Chat template $\mathcal{F}_{\text{template}}$
Require: Number of examples to generate N
1: Initialize dataset $\mathcal{D} \leftarrow \emptyset$
2: {Translate system prompt and few-shot examples}
3: Translate system prompt: $\mathcal{P}_{\text{sys}}^{\ell} \leftarrow \mathcal{M}(\mathcal{P}_{\text{trans}} \oplus \mathcal{P}_{\text{sys}}^{\text{EN}})$
4: Initialize few-shot examples in target language: $\mathbf{E}^{\ell} \leftarrow \emptyset$
5: **for** $(I_i, R_i) \in \mathbf{E}^{\text{EN}}$ **do**
6: $\quad I^{\ell} \leftarrow \mathcal{M}(\mathcal{P}_{\text{trans}} \oplus I_i)$
7: $\quad R^{\ell} \leftarrow \mathcal{M}(\mathcal{P}_{\text{trans}} \oplus R_i)$
8: $\quad \mathbf{E}^{\ell} \leftarrow \mathbf{E}^{\ell} \cup \{(I^{\ell}, R^{\ell})\}$
9: **end for**
10: Construct prompt: $p \leftarrow \mathcal{F}_{\text{template}}(\mathcal{P}_{\text{sys}}^{\ell}, \mathbf{E}^{\ell})$
11: {Synthesize instruction data}
12: **while** $|\mathcal{D}| < N$ **do**
13: $\quad$ Sample instruction: $I \leftarrow \mathcal{M}(p, \text{temperature} = 1.0)$
14: $\quad$ Generate response: $R \leftarrow \mathcal{M}(I, \text{temperature} = 1.0)$
15: $\quad$ {Verify that the generated data is in target language}
16: $\quad$ Detect language: $\hat{\ell} \leftarrow \mathcal{F}_{\text{detect}}(I \oplus R, \text{target} = \ell)$
17: $\quad$ **if** $\hat{\ell} = \ell$ **then**
18: $\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{(I, R)\}$
19: $\quad$ **end if**
20: **end while**
21: **return** $\mathcal{D}$

Vocabulary Simplification for Specific Language

Vocabulary properties vary significantly across languages. To enable efficient processing, we identify a compact subset of tokens from the LLM's vocabulary that sufficiently covers each target language. Our analysis begins with publicly available corpora such as Wikipedia and FineWeb (Penedo et al. 2025), where we tokenize the text and calculate token frequencies. We find that a small subset of high-frequency tokens can cover most of a language's vocabulary (Figure 5a). Notably, Latin-based languages like French, German, and Spanish require larger token sets compared to logographic languages such as Chinese and Japanese. Figure 5b demonstrates that while 8K tokens typically achieve over 95% coverage across languages, the overlap between these token sets remains limited—emphasizing the need for language-specific vocabularies. Certain language groups, particularly Western languages and the Chinese-Japanese pair, exhibit higher overlap due to shared linguistic features (see Appendix for further discussion).

From this analysis, we derive language-specific vocabulary sets sorted by token frequency, which AdaSpec then employs in its adaptive mechanism to streamline the drafter's LM Head.

AdaSpec Adaptive Mechanism

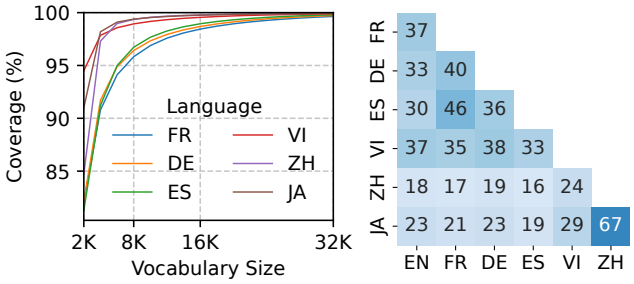
Using the AdaSpec approach to train drafters and simplify vocabularies enables the construction of efficient speculative models tailored to specific languages. However, in practical deployments of LLMs in multilingual environments, it is often impossible to predict the user's language or the language

Algorithm 2: AdaSpec Decoding.

Require: Original Model $\mathcal{M}$; Model vocabulary size $|\mathcal{V}|$; Maximum new tokens N; Set of supported languages $\mathcal{L}$
Require: Vocabulary lists for each language $\mathbf{V} = \{\mathcal{V}_\ell\}_{\ell \in \mathcal{L}}$ {Vocabulary lists are sorted descending by frequency}
List of drafters $\mathcal{D} = \{D_\ell\}_{\ell \in \mathcal{L}}$; Vocabulary size offset z
Require: Model LM head $\mathbf{W}_{\text{LM}} \in \mathbb{R}^{|\mathcal{V}| \times d}$
Require: Window size w; Adaptive frequency u
{Hyperparams for the frequency of adaptive mechanism}
1: Initialize output sequence: $\mathcal{O} \leftarrow \emptyset$
2: Initialize position index: $i \leftarrow 0$
3: {Assign the beginning language, drafter, vocabulary}
4: $\hat{\ell} \leftarrow \mathcal{L}[0]; \quad \mathcal{V}_{\text{cur}} \leftarrow \mathcal{V}_{\hat{\ell}}; \quad s_{\text{cur}} = \mathcal{S}[0]; \quad D_{\text{cur}} \leftarrow \mathcal{D}_{\hat{\ell}}$
5: Set LM head: $\hat{\mathbf{W}}_{\text{LM}} \leftarrow \{\mathbf{W}_{\text{LM}}[v, :] \mid v \in \mathcal{V}_{\text{cur}}[0 : s_{\text{cur}}]\}$
6: {Construct the language mask}
7: $\mathbf{M}_{\text{lang}} \in \{0, 1\}^{|\mathcal{L}| \cdot |\mathcal{S}| \times |\mathcal{V}|}$
8: $\mathbf{M}_{\text{lang}} \leftarrow \mathbf{0}^{|\mathcal{L}| \cdot |\mathcal{S}| \times |\mathcal{V}|}$
9: **for** $l = 0$ to $|\mathcal{L}| - 1$ **do**
10: **for** $s = 0$ to $|\mathcal{S}| - 1$ **do**
11: $\mathcal{V}_{tmp} = \mathbf{V}[l]$
12: **for** $t = 0$ to s **do**
13: $\mathbf{M}_{\text{lang}}[s * |\mathcal{L}| + l, \mathcal{V}_{tmp}[t]] \leftarrow 1$
14: **end for**
15: **end for**
16: **end for**
17: {Construct the offset vector}
18: $\mathbf{m}_{\text{offset}} = [z \times |\mathcal{S}|, z \times (|\mathcal{S}| - 1), \ldots, z \times 1, 0]$
19: **while** $i < N$ **do**
20: {Generate with Speculative Decoding}
21: $\mathcal{T}_{\text{new}} \leftarrow \text{SpecDec}(\mathcal{M}, D_{\text{cur}}, \hat{\mathbf{W}}_{\text{LM}})$
22: $\mathcal{O} \leftarrow \mathcal{O} \oplus \mathcal{T}_{\text{new}}$
23: $i \leftarrow i + |\mathcal{T}_{\text{new}}|$
24: **if** $i \bmod u \neq 0$ **then**
25: {Only run adaptive mechanism every u steps}
26: continue
27: **end if**
28: {Create the mask for the current context}
29: $\mathcal{T}_{\text{ctx}} \leftarrow \mathcal{O}[\max(0, i - w) : i + 1]$
30: $\mathbf{M}_{\text{ctx}} \in \{0, 1\}^{|\mathcal{V}|}$
31: **for** each $t \in \mathcal{T}_{\text{ctx}}$ **do**
32: $\mathbf{M}_{\text{ctx}}[t] \leftarrow 1$
33: **end for**
34: {Compute the vocabulary overlapping}
35: {$\mathbf{M}_{\text{ctx}}$ is broadcast to shape of $\mathbf{M}_{\text{lang}}$ during $\wedge$ operation}
36: Compute overlap: $\mathbf{M}_{\text{overlap}} \leftarrow \mathbf{M}_{\text{lang}} \wedge \mathbf{M}_{\text{ctx}}$
37: Sum overlapping tokens: $\mathbf{m} \leftarrow \sum_{j=1}^{|\mathcal{V}|} \mathbf{M}_{\text{overlap}}[:, j]$
38: Apply the offset vector: $\mathbf{m} = \mathbf{m} + \mathbf{m}_{\text{offset}}$
39: Optimal position: $p_{\max} \leftarrow \arg\max \mathbf{m}$
40: Select language: $\ell_{\text{cur}} = \mathcal{L}[p_{\max} \bmod |\mathcal{L}|]$
41: Select vocabulary size: $s_{\text{cur}} = \mathcal{S}[\left\lfloor \frac{p_{\max}}{|\mathcal{L}|} \right\rfloor]$
42: {Update the current vocabulary, drafter, and LM Head}
43: $\mathcal{V}_{\text{cur}} \leftarrow \mathcal{V}_{\ell_{\text{cur}}}; \quad D_{\text{cur}} \leftarrow \mathcal{D}[\ell_{\text{cur}}]$
44: $\hat{\mathbf{W}}_{\text{LM}} \leftarrow \{\mathbf{W}_{\text{LM}}[v, :] \mid v \in \mathcal{V}_{\text{cur}}[0 : s_{\text{cur}}]\}$
45: **end while**
46: **return** $\mathcal{O}[: N]$

in which the model should respond.

To address this, we propose an adaptive mechanism in AdaSpec Decoding that dynamically selects the appropriate drafter and vocabulary set for LM head simplification



(a) Vocabulary coverage when using most frequent tokens. (b) Vocabulary overlap between languages (size 8K)

Figure 5: Vocabulary analysis of using Llama3 tokens.

based on the ongoing generation context (Algorithm 2). The core idea is to adaptively determine the optimal drafter and vocabulary during decoding, guided by the **most recently generated tokens**.

AdaSpec achieves this through a dedicated *language mask* (lines 6–16), which is a binary matrix of size $|\mathcal{L}| \cdot |\mathcal{S}| \times |\mathcal{V}|$, where $|\mathcal{L}|$ is the number of languages, $|\mathcal{S}|$ is the number of vocabulary size settings, and $|\mathcal{V}|$ is the full vocabulary size. The matrix is organized into $|\mathcal{S}|$ chunks, each containing $|\mathcal{L}|$ rows. Each row corresponds to a specific language and vocabulary size pair, where a value of 1 indicates that a token is included in the vocabulary, and 0 otherwise. Figure 4 illustrates an example with 3 languages and 2 vocabulary sizes. Since larger vocabularies offer broader coverage at the cost of slower LM head computation, we introduce an offset vector controlled by a hyperparameter z to balance this trade-off (line 17).

At each generation step, the most recent w tokens are used to construct a *context mask* (lines 28–33). We then compute the overlap between this context mask and each row of the language mask (lines 34–41) to evaluate the compatibility of each language-vocabulary configuration with the current context. Based on the result, we select the optimal drafter and vocabulary size, and simplify the LM head accordingly. More details of the AdaSpec adaptive mechanism are provided in the Appendix.

Efficient Implementation

Efficient Implementation in C/CUDA. While Python offers rapid prototyping and ease of use, its dynamic typing and interpreted nature can introduce substantial runtime overhead, particularly in latency-critical applications. For example, in EAGLE-2's decoding, which entails numerous short-duration computational tasks, the original PyTorch implementation exhibits high latency due to frequent Python-level function calls (detailed in the Appendix). This finding is consistent with prior work (Zhao et al. 2025). Our C-based implementation provides notable speedups compared to the original Python framework and enhances the performance of the simplified LM head. Additionally, we implement efficient on-the-fly switching of the drafter model and LM head simplification during generation, ensuring that the overall AdaSpec decoding process remains highly efficient.

End-to-End Deployment. ADASPEC supports fully end-to-end deployment. Users only need to specify the language codes they wish to support, without providing any additional data. ADASPEC automatically performs data synthesis, trains the drafter model, analyzes language corpora to extract vocabulary frequency statistics, and integrates all components to produce the final ADASPEC model.

4 Multilingual SpecBench

To enable a rigorous evaluation of speculative decoding in multilingual contexts, we introduce **Multi-SpecBench**, an extension of the original SpecBench (Xia et al. 2024). This benchmark is designed to assess the performance and robustness of decoding methods across seven languages—English, German, French, Spanish, Chinese, Japanese, and Vietnamese—while covering a broad spectrum of real-world tasks. Unlike the original benchmark, which is limited to English-only generation, our multilingual version captures the diversity and complexity of multilingual applications.

Multilingual SpecBench is constructed by collecting and curating samples from existing multilingual evaluation datasets across seven representative tasks: (1) **QA**, including knowledge-intensive and closed-book questions; (2) **RAG**, where retrieved context is combined with generation; (3) **Translation**, translating English text into the target language; (4) **Summarization**, generating summaries of Wikipedia passages; (5) **Code generation**, where input prompts are translated into programming solutions; (6) **Math reasoning**, requiring multi-step generation in structured formats; and (7) **Multi-turn conversation**, evaluating dialogic reasoning, context maintenance. Each task–language pair contains **80 examples** and is aligned to ensure consistency in prompt style and expected output format. The dataset sources used for construction will be listed in the Appendix.

We evaluate decoding methods under two settings: **monolingual**, where evaluation is performed on one language at a time, and **multilingual**, where samples from all seven languages are combined into a single evaluation set. In the multilingual setting, all samples are mixed with each language capped at 15% of the total sample count to maintain balance. This setup better reflects real-world deployment scenarios where inputs from multiple languages may appear unpredictably within the same workload.

5 Experiments

Models. We evaluate ADASPEC on various LLMs: Llama3.2-1B-Instruct (L3.2-1B) and Llama3-8B-Instruct (L3-8B) (Dubey et al. 2024) with 128K vocabulary, and Qwen-2.5-Instruct version 7B (Q2.5-7B) and 14B (Q2.5-14B) (Yang et al. 2025b) with 152K vocabulary.

Datasets. All experiments use the **Multi-SpecBench** benchmark in both monolingual and multilingual settings.

Evaluations Methods. We compare ADASPEC with standard autoregressive decoding and state-of-the-art speculative decoding methods, including EAGLE-2 (Li et al. 2024c), FR-SPEC (Zhao et al. 2025), Prompt Lookup Decoding (PLD), Lookahead (Fu et al. 2024), SAM Decoding (Hu et al. 2024), and Token Recycling (Luo et al. 2025). For all baselines, we adopt hyperparameters reported in their original papers. The evaluation metrics used are as follows:

- **Speedup Ratio ($Spd.$):** The speedup achieved relative to standard autoregressive decoding.
- **Compression Ratio (τ):** The ratio of the total number of autoregressive decoding steps to the number of ADASPEC decoding steps required to generate sequences of the same length.

Implementation. We benchmark both HuggingFace-based Python implementations and our optimized C/CUDA implementations. Unless otherwise specified, experiments are run on a single NVIDIA A100 80GB GPU. Detailed information on the hyperparameters and environment configurations for ADASPEC and all baseline methods can be found in the Appendix.

6 Main Results

Monolingual. Table 1 reports the performance of all evaluated speculative decoding methods across four LLMs and seven languages in the Multilingual Spec-Bench benchmark. Results are shown for both HuggingFace-based Python implementations and our optimized C/CUDA implementations. ADASPEC consistently achieves the highest speedups across all models and languages. It delivers up to **5.62× speedup** over the Python-based autoregressive baseline on the LLaMA-3.2-1B model and up to **1.96× speedup** over the C/CUDA autoregressive implementation on the Qwen-2.5-7B model. In contrast to baselines such as EAGLE-2 and FR-SPEC, which degrade notably in non-English settings, ADASPEC maintains strong performance across all languages, demonstrating its robustness for multilingual inference. Even in English—where baseline methods typically perform best—ADASPEC remains competitive, confirming the effectiveness of its self-generated instruction data and adaptive vocabulary strategies.

Compression ratios (τ) are also highest with ADASPEC. Each draft-and-verify iteration yields on average **2–3 accepted tokens**, significantly outperforming baselines across most languages. These gains are particularly pronounced in non-English languages, where baselines often fail to generalize effectively. Finally, our optimized C/CUDA implementations provide further throughput improvements over Python-based counterparts for all methods, highlighting the value of system-level optimization.

Multilingual. In the multilingual setting, ADASPEC is compared against EAGLE-2 and FR-SPEC. Figure 6 shows throughput results across task categories. ADASPEC consistently outperforms both baselines, with the largest speedups observed in QA and multi-turn conversation tasks that are important for real-world LLM applications. Notably, on the Llama-3.2-1B model, EAGLE-2 and FR-Spec yield lower speedups or even degrade performance compared to vanilla autoregressive decoding. This highlights that speculative decoding, when not carefully adapted, can introduce inefficien-

Model	Method	English Spd.	English τ	German Spd.	German τ	French Spd.	French τ	Spanish Spd.	Spanish τ	Chinese Spd.	Chinese τ	Japanese Spd.	Japanese τ	Vietnamese Spd.	Vietnamese τ	AVG Spd.
L3.2-1B	*Huggingface Implementation in Python*															
	PLD	1.17	1.47	1.43	1.83	1.38	1.76	1.36	1.73	1.96	2.57	2.03	2.70	1.53	1.95	1.55
	EAGLE-2	1.38	3.12	0.98	1.91	1.11	2.04	1.07	2.08	0.98	1.89	0.90	1.71	0.97	1.80	0.72
	Our Optimized Implementation in C															
	Autoreg.	$1.00_{4.22}$	1.00	$1.00_{4.19}$	1.00	$1.00_{4.22}$	1.00	$1.00_{4.21}$	1.00	$1.00_{4.17}$	1.00	$1.00_{4.14}$	1.00	$1.00_{4.19}$	1.00	$1.00_{4.19}$
	EAGLE-2	$0.85_{3.59}$	**3.12**	$0.53_{2.22}$	1.91	$0.58_{2.43}$	2.04	$0.58_{2.44}$	2.08	$0.52_{2.18}$	1.89	$0.47_{1.94}$	1.71	$0.51_{2.12}$	1.80	$0.58_{2.42}$
	FR-SPEC	$1.33_{5.61}$	2.91	$0.84_{3.52}$	1.78	$0.90_{3.81}$	1.90	$0.89_{3.73}$	1.91	$0.70_{2.91}$	1.42	$0.66_{2.73}$	1.33	$0.69_{2.91}$	1.44	$0.86_{3.60}$
	AdaSpec	$\mathbf{1.39_{5.85}}$	2.81	$\mathbf{1.43_{5.98}}$	**2.62**	$\mathbf{1.14_{4.80}}$	**2.08**	$\mathbf{1.20_{5.04}}$	**2.18**	$\mathbf{1.34_{5.59}}$	**2.34**	$\mathbf{1.60_{6.65}}$	**3.18**	$\mathbf{1.30_{5.44}}$	**2.30**	$\mathbf{1.34_{5.62}}$
L3-8B	*Huggingface Implementation in Python*															
	Lookahead	1.32	1.77	1.45	1.66	1.41	1.70	1.40	1.65	1.47	1.74	1.44	1.85	1.45	1.77	1.42
	PLD	1.34	1.57	1.50	1.63	1.47	1.58	1.34	1.55	1.66	1.86	1.74	2.05	1.59	1.78	1.52
	Recycling	2.25	2.77	2.42	2.69	2.53	2.84	2.44	2.85	2.53	**2.91**	2.42	**2.90**	2.47	**2.98**	2.44
	SAMD	2.22	3.64	1.46	2.10	1.64	2.44	1.67	2.55	1.66	2.40	1.64	2.24	1.59	2.34	1.70
	EAGLE-2	1.96	3.62	1.24	2.08	1.51	2.44	1.51	2.61	1.40	2.32	1.16	1.99	1.37	2.30	1.45
	Our Optimized Implementation in C															
	Autoreg.	$1.00_{1.83}$	1.00	$1.00_{1.96}$	1.00	$1.00_{1.97}$	1.00	$1.00_{1.87}$	1.00	$1.00_{1.92}$	1.00	$1.00_{1.83}$	1.00	$1.00_{1.87}$	1.00	$1.00_{1.89}$
	EAGLE-2	$1.84_{3.37}$	**3.62**	$1.15_{2.26}$	2.08	$1.34_{2.63}$	2.44	$1.39_{2.61}$	2.61	$1.27_{2.45}$	2.32	$1.16_{2.13}$	1.99	$1.29_{2.41}$	2.30	$1.35_{2.55}$
	FR-SPEC	$2.00_{3.66}$	3.40	$1.23_{2.41}$	1.89	$1.39_{2.74}$	2.15	$1.41_{2.65}$	2.24	$1.06_{2.04}$	1.52	$1.02_{1.87}$	1.41	$1.16_{2.16}$	1.61	$1.33_{2.51}$
	AdaSpec	$\mathbf{2.11_{3.86}}$	3.61	$\mathbf{1.83_{3.59}}$	**2.71**	$\mathbf{1.96_{3.86}}$	**2.93**	$\mathbf{1.90_{3.56}}$	**2.86**	$\mathbf{1.78_{3.43}}$	**2.49**	$\mathbf{1.91_{3.49}}$	**2.79**	$\mathbf{1.91_{3.56}}$	**2.68**	$\mathbf{1.91_{3.62}}$
Q2.5-7B	*Huggingface Implementation in Python*															
	PLD	1.22	1.39	1.32	1.49	1.27	1.44	1.31	1.48	1.37	1.53	1.54	1.70	1.42	1.59	1.35
	EAGLE-2	1.97	3.46	1.09	1.71	1.28	2.06	1.22	1.97	1.20	1.90	0.91	1.42	1.15	1.86	1.26
	Our Optimized Implementation in C															
	Autoreg.	$1.00_{1.76}$	1.00	$1.00_{1.76}$	1.00	$1.00_{1.76}$	1.00	$1.00_{1.76}$	1.00	$1.00_{1.77}$	1.00	$1.00_{1.79}$	1.00	$1.00_{1.77}$	1.00	$1.00_{1.77}$
	EAGLE-2	$1.59_{2.79}$	**3.46**	$0.88_{1.54}$	1.71	$1.04_{1.83}$	2.06	$0.99_{1.74}$	1.97	$0.96_{1.70}$	1.90	$0.73_{1.30}$	1.42	$0.94_{1.66}$	1.86	$1.02_{1.80}$
	FR-SPEC	$1.98_{3.48}$	3.23	$1.12_{1.97}$	1.66	$1.28_{2.25}$	1.91	$1.25_{2.20}$	1.88	$0.94_{1.66}$	1.35	$0.88_{1.56}$	1.30	$0.99_{1.76}$	1.46	$1.21_{2.13}$
	AdaSpec	$\mathbf{2.02_{3.55}}$	3.16	$\mathbf{1.85_{3.25}}$	**2.69**	$\mathbf{1.81_{3.19}}$	**2.58**	$\mathbf{2.00_{3.53}}$	**2.90**	$\mathbf{1.67_{2.95}}$	**2.36**	$\mathbf{2.18_{3.89}}$	**3.15**	$\mathbf{2.17_{3.83}}$	**3.08**	$\mathbf{1.96_{3.46}}$
Q2.5-14B	*Huggingface Implementation in Python*															
	PLD	1.28	1.37	1.40	1.51	1.32	1.44	1.34	1.48	1.41	1.52	1.61	1.74	1.52	1.60	1.41
	EAGLE-2	2.68	3.84	1.75	2.26	2.00	2.64	2.03	2.66	1.80	2.38	1.53	1.91	1.84	2.23	1.95
	Our Optimized Implementation in C															
	Autoreg.	$1.00_{1.53}$	1.00	$1.00_{1.58}$	1.00	$1.00_{1.56}$	1.00	$1.00_{1.56}$	1.00	$1.00_{1.54}$	1.00	$1.00_{1.58}$	1.00	$1.00_{1.60}$	1.00	$1.00_{1.57}$
	EAGLE-2	$1.36_{2.08}$	**3.84**	$1.04_{1.64}$	2.26	$1.17_{1.83}$	**2.64**	$1.15_{1.79}$	2.66	$1.08_{1.67}$	**2.38**	$0.91_{1.43}$	1.91	$0.98_{1.57}$	2.23	$1.10_{1.72}$
	FR-SPEC	$\mathbf{1.63_{2.49}}$	3.59	$1.18_{1.86}$	2.00	$1.31_{2.06}$	2.31	$1.25_{1.96}$	2.28	$0.93_{1.43}$	1.38	$0.88_{1.39}$	1.33	$0.98_{1.58}$	1.50	$1.17_{1.82}$
	AdaSpec	$1.59_{2.44}$	3.30	$\mathbf{1.55_{2.45}}$	**2.84**	$\mathbf{1.37_{2.14}}$	2.33	$\mathbf{1.42_{2.21}}$	**2.67**	$\mathbf{1.35_{2.08}}$	2.36	$\mathbf{1.48_{2.34}}$	**2.79**	$\mathbf{1.36_{2.19}}$	**2.52**	$\mathbf{1.45_{2.26}}$

Table 1: Performance of speculative decoding methods in a monolingual setting and two implementations (Python/C) for each model, measured in speedup ratio (Spd) and compression ratio (τ). For C implementation, the speedup value is the speedup ratio compared to Autoregressive implemented in **C**. The value in blue indicate the speedup ratio compared to **Python** implementation of Autoregressive Decoding.

cies rather than improvements, especially in smaller models or multilingual contexts.

7 Analysis

Ablation Study. We conduct a detailed ablation study in the monolingual setting to evaluate the contribution of each AdaSpec component (Table 2). Using LLaMA3-8B-Instruct, we find that removing the target-language-trained drafter significantly reduces speedup—from **1.83**× to **1.30**× in German and from **1.91**× to **1.25**× in Japanese. This demonstrates that language-specific instruction tuning is critical for high token acceptance and speedup. Similarly, replacing the target-language vocabulary with the default full vocabulary lowers speedup to **1.58**× (German) and **1.66**× (Japanese). Finally, disabling AdaSpec's adaptive vocabulary size selection—replacing it with a fixed-size vocabulary—also leads to a measurable drop in speedup, confirming the benefits of dynamically adjusting vocabulary size for different languages. Despite these degradations, all variants still outperform prior methods, highlighting the complementary strength of AdaSpec's components.

Method	German Spd.	German τ	Japanese Spd.	Japanese τ
EAGLE-2	1.15	2.08	1.16	1.99
FR-Spec	1.23	1.89	1.02	1.41
AdaSpec (ours)	**1.83**	2.71	**1.91**	2.79
- w/o target drafter	1.30	1.96	1.25	1.86
- w/o target vocab	1.58	**2.79**	1.66	**2.94**
- w/o adaptive vocab size	1.71	2.76	1.77	2.88

Table 2: Ablation study of AdaSpec components in monolingual settings (LLaMA3-8B-Instruct).

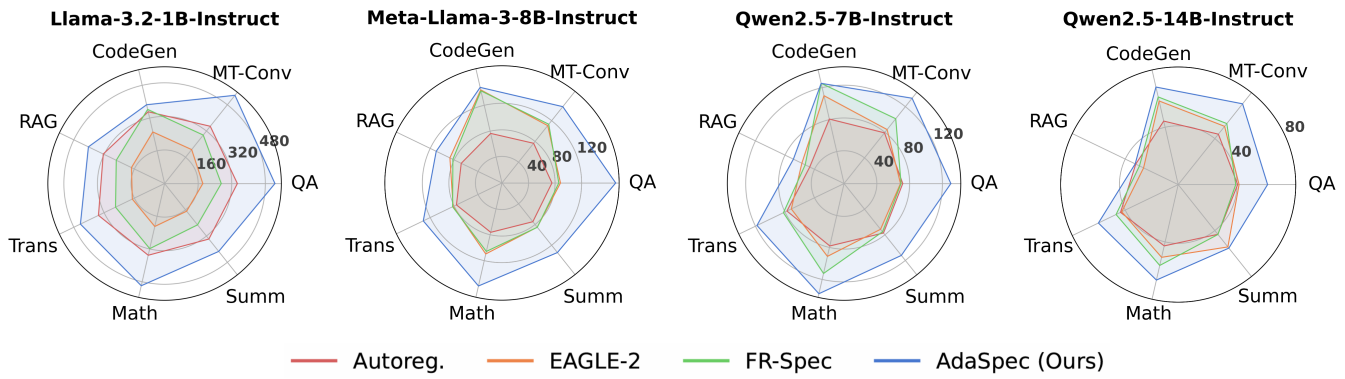


Figure 6: Throughput comparison in the multilingual setting for four LLMs under different speculative decoding methods across seven tasks: Code Generation (**CodeGen**), Multi-turn Conversation (**MT-Conv**), Question Answering (**QA**), Summarization (**Summ**), Math Reasoning (**Math**), Translation (**Trans**), and Retrieval-Augmented Generation (**RAG**).

Impact of Window Size, Step Size, and Offset. We analyze the effect of the **window size** (w), **step size** (u), and **offset** (z) in Algorithm 2 on AdaSpec's performance in multilingual settings (Table 3). Increasing w from 5 to 15 with fixed $u = 1$ and $z = 0$ (Tag 3–5) provides positive returns: throughput rises slightly (135.22 $\rightarrow$ 136.70). Introducing a larger step size ($u = 3$) and non-zero offset ($z = 1$) (Tag 6–8) further boosts throughput compared to the baseline configuration at $w = 10$ (138.30 vs. 136.30). The best configuration, ($w = 10$, $u = 5$, $z = 2$) (Tag 9), achieves the highest throughput (139.93), speedup (1.87$\times$), and compression ratio ($\tau = 2.82$), striking an balance between token acceptance and efficiency. Notably, even the lowest-performing AdaSpec setting (Tag3) outperforms both the autoregressive baseline (Tag1) and EAGLE-2 (Tag2) by a wide margin, demonstrating the advantage of adaptive parameterization in speculative decoding for multilingual scenarios.

Tag	SETTING (w,u,z)	Throughput	Spd.	τ
(1)	Autoreg.	74.78	1.00	1.00
(2)	EAGLE-2	100.91	1.35	2.36
(3)	(5, 1, 0)	135.22	1.81	2.75
(4)	(10, 1, 0)	136.30	1.82	2.78
(5)	(15, 1, 0)	136.70	1.83	2.81
(6)	(5, 3, 1)	136.95	1.83	2.71
(7)	(10, 3, 1)	138.30	1.85	2.75
(8)	(15, 3, 1)	138.07	1.85	2.76
(9)	(10, 5, 2)	**139.93**	**1.87**	**2.82**
(10)	(15, 5, 2)	138.91	1.86	2.75

Table 3: Impact of window size, step size, and vocabulary size offset on LLaMA3-8B-Instruct in multilingual settings.

8 Related Works

Speculative decoding has emerged as a promising approach for accelerating LLM inference by generating multiple tokens in parallel without compromising output quality (Chen et al. 2023b; Leviathan, Kalman, and Matias 2023). It follows a draft-and-verify paradigm, where a drafter proposes multiple tokens in parallel, and the base LLM verifies their correctness. Existing approaches fall into two categories: training-free and training-based. Training-free methods (He et al. 2024; Li et al. 2024a; Ou, Chen, and Tian 2024; Hu et al. 2024; Le, Do, and Nguyen 2025) use retrieval or internal model knowledge to select candidate tokens without additional training. For instance, REST (He et al. 2024) and ANLPD (Ou, Chen, and Tian 2024) leverage external stores, while Lookahead (Fu et al. 2024) and Jacobi-style decoding (Santilli et al. 2023) operate directly on the model's predictions. Training-based methods, though requiring extra supervision, often achieve higher speedups. Medusa (Cai et al. 2024) employs a simple MLP-based drafter, while EAGLE (Li et al. 2024d,c) uses Transformer layers to generate more accurate drafts. FR-SPEC (Zhao et al. 2025) builds on EAGLE by pruning the vocabulary to reduce drafting costs. However, these methods have primarily focused on English, and their effectiveness in multilingual scenarios remains largely unexplored.

Recent efforts have begun to address multilingual acceleration. Hong, Lee, and Cho (2024) proposed language-specific vocabulary heads with fine-tuning, while Yi et al. (2024) introduced a two-stage drafter training approach using general corpora followed by task-specific data.

9 Conclusions

We introduce AdaSpec, a multilingual speculative decoding framework that trains language-specific drafters on self-synthesized instruction data and simplifies the LM head using language-aware vocabulary sets. By generating training data in the target language using the LLM itself, AdaSpec enables effective drafter training even for low-resource languages. Key vocabularies are identified through token frequency analysis on language-specific corpora. An adaptive mechanism dynamically selects the optimal drafter and vocabulary set during generation.

References

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

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. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org. Place: Vienna, Austria.

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

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

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Fu, Y.; Bailis, P.; Stoica, I.; and Zhang, H. 2024. Break the sequential dependency of LLM inference using LOOKAHEAD DECODING. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org. Place: Vienna, Austria.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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.

Hong, J.; Lee, G.; and Cho, J. 2024. Accelerating Multilingual Language Model for Excessively Tokenized Languages. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics: ACL 2024*, 11095–11111. Bangkok, Thailand: Association for Computational Linguistics.

Hu, Y.; Wang, K.; Zhang, X.; Zhang, F.; Li, C.; Chen, H.; and Zhang, J. 2024. SAM Decoding: Speculative Decoding via Suffix Automaton. arXiv:2411.10666.

Le, N.-K.; Do, T. D.; and Nguyen, L.-M. 2025. SPECTRA: Faster Large Language Model Inference with Optimized Internal and External Speculation. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 14015–14034. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-251-0.

Leviathan, Y.; Kalman, M.; and Matias, Y. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org. Place: Honolulu, Hawaii, USA.

Li, M.; Chen, X.; Holtzman, A.; Chen, B.; Lin, J.; Yih, W.-t.; and Lin, X. V. 2024a. Nearest Neighbor Speculative Decoding for LLM Generation and Attribution. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024b. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 7421–7432. Miami, Florida, USA: Association for Computational Linguistics.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024c. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 7421–7432. Miami, Florida, USA: Association for Computational Linguistics.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024d. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2025. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. arXiv:2503.01840.

Luo, X.; Wang, Y.; Zhu, Q.; Zhang, Z.; Zhang, X.; Yang, Q.; and Xu, D. 2025. Turning Trash into Treasure: Accelerating Inference of Large Language Models with Token Recycling. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 6816–6831. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-251-0.

Miao, X.; Oliaro, G.; Zhang, Z.; Cheng, X.; Wang, Z.; Zhang, Z.; Wong, R. Y. Y.; Zhu, A.; Yang, L.; Shi, X.; Shi, C.; Chen, Z.; Arfeen, D.; Abhyankar, R.; and Jia, Z. 2024. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, 932–949. New York, NY, USA: Association for Computing Machinery. ISBN 979-8-4007-0386-7. Event-place: La Jolla, CA, USA.

Ou, J.; Chen, Y.; and Tian, P. 2024. Lossless Acceleration of Large Language Model via Adaptive N-gram Parallel Decoding. In Yang, Y.; Davani, A.; Sil, A.; and Kumar, A., eds., *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, 10–22. Mexico City, Mexico: Association for Computational Linguistics.

Penedo, G.; Kydlíček, H.; Sabolčec, V.; Messmer, B.; Foroutan, N.; Kargaran, A. H.; Raffel, C.; Jaggi, M.; Werra, L. V.; and Wolf, T. 2025. FineWeb2: One Pipeline to Scale Them All – Adapting Pre-Training Data Processing to Every Language. _eprint: 2506.20920.

Santilli, A.; Severino, S.; Postolache, E.; Maiorca, V.; Mancusi, M.; Marin, R.; and Rodola, E. 2023. Accelerating Transformer Inference for Translation via Parallel Decoding. In Rogers, A.; Boyd-Graber, J.; and Okazaki, N., eds., *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 12336–12355. Toronto, Canada: Association for Computational Linguistics.

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

Xia, H.; Yang, Z.; Dong, Q.; Wang, P.; Li, Y.; Ge, T.; Liu, T.; Li, W.; and Sui, Z. 2024. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of the Association for Computational Linguistics: ACL 2024*, 7655–7671. Bangkok, Thailand: Association for Computational Linguistics.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; Lin, H.; Yang, J.; Tu, J.; Zhang, J.; Yang, J.; Yang, J.; Zhou, J.; Lin, J.; Dang, K.; Lu, K.; Bao, K.; Yang, K.; Yu, L.; Li, M.; Xue, M.; Zhang, P.; Zhu, Q.; Men, R.; Lin, R.; Li, T.; Tang, T.; Xia, T.; Ren, X.; Ren, X.; Fan, Y.; Su, Y.; Zhang, Y.; Wan, Y.; Liu, Y.; Cui, Z.; Zhang, Z.; and Qiu, Z. 2025b. Qwen2.5 Technical Report. arXiv:2412.15115.

Yi, E.; Kim, T.; Jeung, H.; Chang, D.-S.; and Yun, S.-Y. 2024. Towards Fast Multilingual LLM Inference: Speculative Decoding and Specialized Drafters. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 10789–10802. Miami, Florida, USA: Association for Computational Linguistics.

Zhang, L.; Wang, X.; Huang, Y.; and Xu, R. 2025. Learning Harmonized Representations for Speculative Sampling. In *The Thirteenth International Conference on Learning Representations*.

Zhao, W.; Pan, T.; Han, X.; Zhang, Y.; Sun, A.; Huang, Y.; Zhang, K.; Zhao, W.; Li, Y.; Wang, J.; et al. 2025. FR-Spec: Accelerating Large-Vocabulary Language Models via Frequency-Ranked Speculative Sampling. *arXiv preprint arXiv:2502.14856*.