OPTISPEC: Device-Calibrated Speculative Decoding for Consistent Large Language Model Speedup Across Hardware

Anonymous ACL submission

Abstract

Speculative decoding accelerates large language models (LLMs) via a draft-and-verify paradigm, but end-to-end gains vary widely across hardware because verification cost grows with draft length. We introduce OPTISPEC, a training-free, lossless, and environment-aware speculative decoding framework that maximizes throughput by matching speculation to device characteristics. OPTISPEC performs a brief device-aware calibration to learn the draft size that optimizes the acceptance–time ratio, then constructs a dynamic draft tree whose nodes are ranked by confidence scores derived from the verifier's token probabilities and refined through tree expansion with threshold-based pruning. In experiments, we evaluate OPTISPEC across diverse generation tasks in seven languages with multiple LLMs and hardware environments, showing consistent performance gains and speedups of up to $2.6\times$. Our code is publicly available.

1 Introduction

Generating long sequences with low latency using Large Language Models (LLMs) is a critical requirement, especially in the recent trends of inference-time scaling (Zhang et al., 2025a; Qu et al., 2025). Current LLMs rely on autoregressive decoding (Touvron et al., 2023; Bai et al., 2023; Jiang et al., 2023; OpenAI et al., 2024), which suffers from inefficiency because it generates text one token at a time. This results in generation time scaling linearly with the sequence length and underutilizes the parallel processing capabilities of modern GPUs. A widely studied approach to mitigate this issue is speculative decoding (Chen et al., 2023; Leviathan et al., 2023), which follows a *guess-and-verify* paradigm. In this approach, a smaller LLM (draft model) (Chen et al., 2023; Leviathan et al., 2023; Miao et al., 2024; Sun et al., 2023; Zhou et al., 2024; Cai et al., 2024) or the original LLM trained in a specialized manner (self-speculative decoding) (Elhoushi et al., 2024; Liu et al., 2024a; Yang et al., 2024; Zhang et al., 2024; Li et al., 2024b) predicts multiple tokens in advance. The original LLM then verifies these predictions in parallel, improving efficiency. However, these approaches require additional training, which demands substantial computational resources and may degrade the original model's capabilities.

Another line of research explores *training-free* speculative decoding, which predicts subsequent tokens without additional training or model modification. This makes such methods practical for off-the-shelf deployment. They generate speculative tokens directly from the LLM's own predictions or from external sources (Le et al., 2025; Fu et al., 2024; Ou et al., 2024; He et al., 2024; Li et al., 2024a; Yang et al., 2023) (Figure 1, Left). At each decoding step, draft tokens are retrieved from an n-gram store to build a draft tree, verified in parallel within a single forward pass, and then used to update the store based on the LLM's output probabilities. Despite growing interest in training-free speculative decoding, current methods face three key limitations. First, they rely on fixed speculation templates (e.g., preset draft lengths or static n-gram expansion rules) (Le et al., 2025; Luo et al., 2025; Fu et al., 2024; He et al., 2024), which ignore device-specific verification costs. A draft size optimal for data-center GPUs (e.g., A100) may degrade performance on consumer GPUs (e.g., RTX 3090) due to increased latency from longer inputs, leading to inconsistent speedups across hardware. Second, existing approaches treat all n-grams equally, lacking a mechanism to assess their quality, which limits the ability to selectively retain high-quality nodes in the draft tree. Third, current tree expansion strategies in state-of-the-art methods remain suboptimal, leaving room for further improvement.

To address these issues, we propose OPTISPEC (Figure 1, Right), a training-free and device-

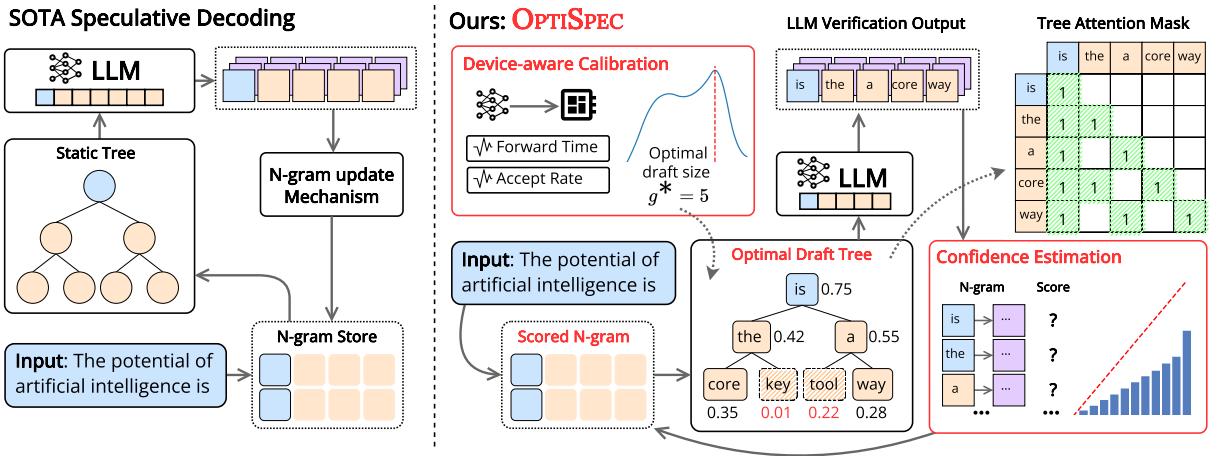

Figure 1: Overview of OptiSpec compared with other SOTA approaches. Newly introduced component is highlighted in red. blue boxes indicate the input tokens, orange boxes indicate the draft tokens, purple boxes indicate top next-token predictions of the LLM.

calibrated speculative decoding framework for robust, plug-and-play LLM acceleration. Upon deployment, OptiSpec first calibrates the hardware to measure forward latency and acceptance rate, deriving the optimal draft size $g^\star$. Using this value, it constructs an adaptive draft tree with $g^\star$ nodes per decoding step. Next, it introduces a confidence estimation mechanism that assigns quality scores to n-grams based on next-token probabilities, enabling selective retention of top-ranked nodes. Finally, OptiSpec enhances tree expansion through broader first-level exploration and score-based filtering, ensuring both efficiency and quality in draft generation. Empirical results on seven generation tasks from Spec-Bench (Xia et al., 2024), evaluated with widely used LLMs — LLaMA3-8B-Instruct (Dubey et al., 2024) and Qwen3-8/14B (Yang et al., 2025) — show that OptiSpec surpasses other training-free speculative decoding methods, achieving speedups of up to 2.6×. To further assess multilingual performance, we extend Spec-Bench to Multi-SpecBench, a multilingual benchmark covering seven languages (English, German, French, Spanish, Chinese, Japanese, and Vietnamese) across the original seven tasks. Results remain consistent, confirming the strong performance of our method across multilingual settings. We publicly release our code and benchmark. In summary, the key contributions of this paper are:

- We propose OptiSpec, a *training-free and device-calibrated* speculative decoding framework that enables robust, plug-and-play acceleration of large language models across heterogeneous hardware.
- OptiSpec introduces (1) a *hardware calibration mechanism* to derive the optimal draft size $g^\star$ for each device, ensuring consistent speedup; (2) a *confidence-guided tree construction strategy* that scores and filters n-grams based on model probabilities; (3) an *improved tree expansion techniques* to enhance the quality and efficiency of draft generation.
- Extensive multilingual and multi-hardware evaluation — we extend Spec-Bench to Multi-SpecBench and demonstrate speedups of up to 2.6× with robust gains on both data-center and consumer GPUs.

2 Preliminaries

2.1 Autoregressive Decoding in LLMs

Given an input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_s)$ of length s, and a slice of length m as $\mathbf{x}_{1:m} = (x_1, x_2, \ldots, x_m)$, the output of an LLM represents a probability distribution over the next token. The probability of generating the s-th token, conditioned on all preceding tokens, is given by $P_M(x_s \mid \mathbf{x}_{1:s-1})$. The next token x_s is sampled from this distribution using methods such as greedy, top-k, or top-p sampling (Kool et al., 2020; Holtzman et al., 2020). For greedy sampling, the next token is selected as $x_s = \text{argmax}\ P_M(x_s \mid \mathbf{x}_{1:s-1})$. Consequently, the LLM generates an output sequence $(y_1, y_2, \ldots, y_m)$ of length m autoregressively, where each token y_i is computed as $y_i = \text{argmax}\ P_M(y_i \mid y_{1:i-1}, \mathbf{x})$.

2.2 Speculative Decoding.

Speculative decoding employs a *guess-and-verify* paradigm where multiple candidate token sequences are drafted and verified in a single decoding step. Using tree attention (Miao et al., 2024), the model can verify G draft sequences $\tilde{Y} = \tilde{y}^{(1)}, \tilde{y}^{(2)}, \ldots, \tilde{y}^{(G)}$ of length K simultaneously. Given prompt $\mathbf{x}$, the drafting method generates $\tilde{Y}$, and the LLM computes the true output tokens $(y'_1, y'_2, \ldots, y'_K)$ in parallel. Let h be the maximum number of tokens matched across all drafts; then $h + 1$ tokens are accepted in one forward pass. Algorithm 2 formalizes this process for greedy sampling.

2.3 Training-free N-gram Strategies

Several prior works employ either a trained drafter model (Chen et al., 2023; Leviathan et al., 2023; Miao et al., 2024; Sun et al., 2023; Zhou et al., 2024; Cai et al., 2024) or modify the original LLM in a specialized manner (self-speculative decoding) (Elhoushi et al., 2024; Liu et al., 2024a; Yang et al., 2024; Zhang et al., 2024; Li et al., 2024b) to predict multiple tokens in advance. While effective, these approaches require costly training and may alter the inherent capabilities of the original LLM.

As an alternative to methods requiring auxiliary models or fine-tuning, training-free n-gram strategies generate draft tokens directly from stored token sequences without additional training. Common approaches use n-gram stores with different retrieval strategies: one-to-many (Fu et al., 2024; He et al., 2024), many-to-one (Ou et al., 2024), one-to-one (Luo et al., 2025), or a combination of these strategy (Le et al., 2025; Hu et al., 2025). For a detailed discussion of these strategies, see Appendix B.

3 Methodology

OPTISPEC is designed to optimize speculative decoding across diverse GPU environments. As illustrated in Figure 1, it follows the standard speculative decoding pipeline, where an n-gram store is used to construct a draft tree, the draft tokens are verified by the target LLM, and the resulting output probabilities are used to update the store. Beyond this baseline, OPTISPEC introduces three key modules: *Device-aware Calibration*, *Confidence Score Estimation*, and *Improved Tree Expansion*.

The *Device-aware Calibration* module measures the hardware's forward latency and acceptance rate to determine the optimal draft size $g^\star$. Once $g^\star$ is obtained, the *Confidence Score Estimation* module assigns quality scores to n-grams, enabling the selection of the top $g^\star$ nodes in the draft tree. Finally, the *Improved Tree Expansion* module enhances tree construction through broader first-level exploration and score-based filtering, improving both efficiency and generation quality. The following subsections describe each component in detail.

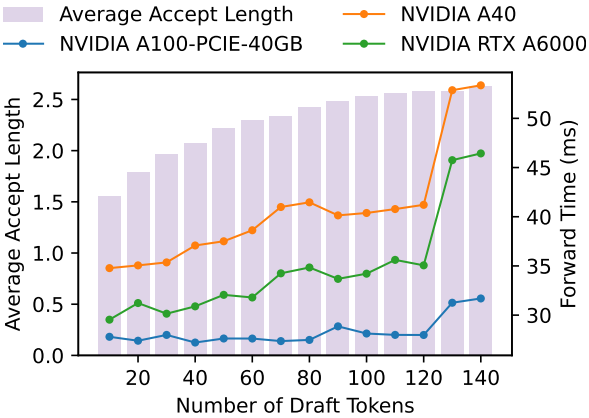

Figure 2: Average acceptance length and forward time by the number of draft tokens (Llama3-8B-Instruct).

Device-aware Calibration. In speculative decoding, each decoding step consists of two phases: *drafting* and *verification*. During drafting, a small draft model—or the target model itself under speculative sampling—generates multiple candidate tokens as potential continuations. In verification, the target model evaluates these candidates in parallel within a single forward pass, accepting as many as possible while ensuring correctness.

The efficiency of speculative decoding is governed by two coupled factors: (i) the average acceptance length of drafted tokens and (ii) the computational cost of the verification step. Increasing the number of drafted tokens generally raises the acceptance length but also the verification cost due to longer input sequences. Hence, performance depends on balancing these opposing effects. Empirically, the average acceptance length remains relatively stable across environments, while verification speed varies substantially with hardware characteristics such as GPU architecture and parallelization efficiency (Figure 2). Consequently, the optimal number of drafted tokens is device-dependent, necessitating hardware-specific calibration to maximize decoding speedup. Consider one decoding step in speculative decoding. Let s denote

the time required for a single forward step of the target model, and let τ denote the average acceptance length of the drafted tokens. The throughput of speculative decoding, measured in tokens generated per unit time, can be expressed as:

$$\text{Throughput} = \frac{\text{Number of tokens generated}}{\text{Total generation time}} \quad (1)$$

$$= \frac{\tau \cdot N_{\text{step}}}{N_{\text{step}} \cdot s} = \frac{\tau}{s} \quad (2)$$

where N_{step} is the number of forward steps. Thus, optimizing throughput reduces to optimizing the ratio τ/s. This optimization can be carried out by running a small number of warm-up steps for calibration in the target deployment environment. Both τ and s are strongly influenced by the number of drafted tokens. Let g denote the number of drafted tokens (draft size). Suppose we perform m calibration runs with varying draft sizes $g_1, g_2, \ldots, g_m$. For each g_i, we measure the forward time s_i and average acceptance length τ_i, yielding two datasets: $S = \{(g_1, s_1), (g_2, s_2), \ldots, (g_m, s_m)\}$, $T = \{(g_1, \tau_1), (g_2, \tau_2), \ldots, (g_m, \tau_m)\}$. We then fit simple regression models to capture the dependency of forward time and average acceptance length on g:

$$f_{\text{time}}(g) \approx s, \quad f_{\text{accept}}(g) \approx \tau. \quad (3)$$

The throughput function can then be approximated as:

$$f_{\text{speed}}(g) = \frac{f_{\text{accept}}(g)}{f_{\text{time}}(g)}. \quad (4)$$

Finally, the optimal draft size can be obtained by solving the following optimization problem:

$$g^\star = \arg\max_g \; f_{\text{speed}}(g). \quad (5)$$

In practice, this optimization can be efficiently solved using global optimization methods such as *Differential Evolution* (Storn and Price, 1997), which are well-suited for non-convex search spaces. The calibration procedure is shown in Algorithm 1, lines 3–11. The detailed implementation procedure is provided in Appendix G.

Tree Node Confidence Score Estimation. After determining the optimal draft size $g^\star$, we require a strategy to select $g^\star$ draft tokens from the draft tree. Unlike prior methods using static trees, we construct a dynamic draft tree where each node is assigned a *confidence score* reflecting its likelihood of being accepted during verification. This score is computed using the target model's token probability distribution. The estimation process is outlined in Algorithm 1, line 38. As illustrated in Figure 3, the confidence score correlates strongly with token acceptance rates, enabling effective draft selection. After each forward pass, generated tokens and their confidence scores are stored in the n-gram store for use in subsequent decoding steps.

Algorithm 1 OPTISPEC Decoding Process

Require: Input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, model P_M, environment $\mathcal{E}$, vocabulary size V, maximum number of new tokens m, maximum tree depth d, number of candidates per level k, confidence threshold r

1: Initialize n-gram token store $\mathcal{S}_{\text{tok}} \leftarrow \emptyset$ (shape $[V, k]$)
2: Initialize n-gram confidence store $\mathcal{S}_{\text{con}} \leftarrow \emptyset$ (shape $[V, k]$)
3: {Hardware calibration to obtain optimal draft size $g^\star$}
4: $S, T \leftarrow \text{CalibrateRun}(P_M, \mathcal{E})$
5: {S and T samples has the following form}
6: $S = \{(g_1, s_1), (g_2, s_2), \ldots, (g_m, s_m)\}$
7: $T = \{(g_1, \tau_1), (g_2, \tau_2), \ldots, (g_m, \tau_m)\}$
8: $f_{\text{time}}(g) \leftarrow \text{LearnRegressionModel}(S)$
9: $f_{\text{accept}}(g) \leftarrow \text{LearnRegressionModel}(T)$
10: $f_{\text{speed}}(g) = f_{\text{accept}}(g)/f_{\text{time}}(g)$
11: $g^\star = \arg\max_g f_{\text{speed}}(g)$
12: {Generation loop}
13: $t \leftarrow n + 1$
14: **while** $t \leq n + m$ **do**
15: {Build optimal draft tree}
16: Initialize draft tree $\mathcal{T}_{\text{tok}} \leftarrow \emptyset$, confidence $\mathcal{T}_{\text{con}} \leftarrow \emptyset$
17: Current leaf nodes $\mathcal{L} \leftarrow \{x_t\}$
18: {Expand first-level draft tree}
19: $\mathcal{T}_{\text{tok}}[0] \leftarrow \mathcal{S}_{\text{tok}}[x_t] \oplus \mathcal{C}_{\text{tok}}$
20: $\mathcal{T}_{\text{con}}[0] \leftarrow \mathcal{S}_{\text{con}}[x_t] \oplus \mathcal{C}_{\text{con}}$
21: {Expand subsequent levels}
22: **for** $i = 1$ **to** d **do**
23: **for** $u \in \mathcal{L}$ **do**
24: $\mathcal{T}_{\text{tok}}[i] \leftarrow \mathcal{T}_{\text{tok}}[i] \oplus \mathcal{S}_{\text{tok}}[u]$
25: $\mathcal{T}_{\text{con}}[i] \leftarrow \mathcal{T}_{\text{con}}[i] \oplus \mathcal{S}_{\text{con}}[u]$
26: **end for**
27: $\mathcal{L} \leftarrow \text{TopK}(\mathcal{T}_{\text{con}}[i], k)$
28: Filter $\mathcal{L}$ to exclude nodes with confidence $< r$
29: **end for**
30: Select optimal draft $\mathcal{T} \leftarrow$ top-$g^\star$ nodes from $\mathcal{T}_{\text{tok}}$
31: {Parallel forward pass in LLM}
32: Obtain output distributions $P_M(\mathcal{T})$
33: {Verification}
34: Draft sequences $\mathcal{G} \leftarrow \text{GetSequences}(\mathcal{T})$
35: hits $\leftarrow \text{VerificationFunction}(\mathbf{x}, P_M, \mathcal{G})$
36: $\mathbf{x} \leftarrow \mathbf{x} \oplus$ hits
37: $t \leftarrow t + |\text{hits}|$
38: {Score estimation and n-gram store update}
39: $\mathbf{O} \leftarrow P_M(\text{flatten}(\mathcal{T}))$ ▷ Shape $[g^\star, V]$
40: $(\mathbf{K}, \text{score}) \leftarrow \text{TopK}(\mathbf{O}, k)$ ▷ Shape $[g^\star, k]$
41: **for** each node $u \in \mathcal{T}$ **do**
42: $\mathcal{S}_{\text{tok}}[u] \leftarrow \mathbf{K}[u]$
43: $\mathcal{S}_{\text{con}}[u] \leftarrow \text{score}[u]$
44: **end for**
45: {Cache first-level candidates for next step}
46: $\mathcal{C}_{\text{tok}} \leftarrow \mathbf{K}[x_{t-1}]$
47: $\mathcal{C}_{\text{con}} \leftarrow \text{score}[x_{t-1}]$
48: **end while**
49: **Output:** Generated tokens $\mathbf{y} = \mathbf{x}_{n+1:n+m}$

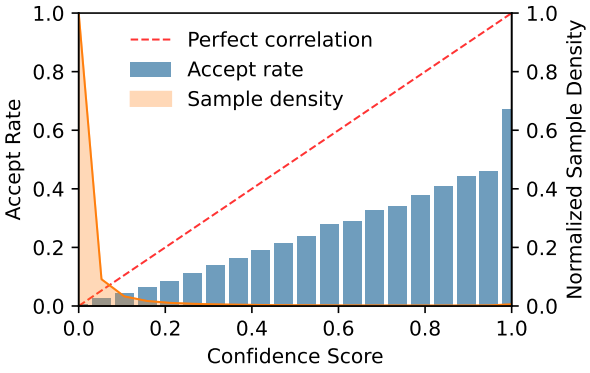

Figure 3: Acceptance rate at different confidence scores, measured in Llama3-8B-Instruct. Sample density indicates the probability that a node with a specific confidence score appears in the draft tree.

Tree Expansion Mechanism. We introduce two techniques to enhance the draft tree's quality: broadening the first-level expansion and filtering low-quality nodes. First, we recognize that the initial level of the draft tree critically impacts performance, as the rejection of a first-level node nullifies its entire subtree. To generate more high-quality candidates at this level, we leverage the insight that large language models can effectively predict the "next-next" token (Liu et al., 2025). Consequently, we cache the likely successors of the token preceding the current root (Alg. 1, line 45) and use them to expand the first level of the draft tree in the subsequent iteration (Alg. 1, line 18). Second, since draft trees often contain numerous nodes with low confidence scores that slow decoding despite low acceptance rates (Sample Density in Figure 3), we implement a threshold-based filtering mechanism. This prunes low-quality nodes during tree expansion, improving efficiency (Alg. 1, line 28).

4 Experiments

Datasets. We evaluate our methods on Spec-Bench (Xia et al., 2024), a widely used benchmark and unified evaluation platform for speculative decoding. Spec-Bench covers seven tasks: multi-turn conversation, translation, summarization, question answering, mathematical reasoning, and retrieval-augmented generation, each with 80 samples per subtask. In addition, we include the HumanEval benchmark (Chen et al., 2021), which focuses on code generation and contains 80 entries.

Since Spec-Bench is limited to English, we extend it to create the Multilingual Spec-Bench (Multi-SpecBench). Unlike the original benchmark, Multi-SpecBench spans seven languages: English, German, French, Spanish, Chinese, Japanese, and Vietnamese. For each language, we maintain the same eight task categories as the English version, with 80 samples per task. This multilingual extension enables a more comprehensive analysis of speculative decoding across diverse linguistic settings. The dataset construction process for Multi-SpecBench is detailed in Appendix C.

Models. We select Llama-3-8B-Instruct (L3-8B) (Dubey et al., 2024), Qwen-3-8B (Q3-8B), and Qwen-3-14B (Q3-14B) (Yang et al., 2025) as the base language models for our experiments. These models are recent, widely adopted, and representative of state-of-the-art large language models.

Testbed. To evaluate the environment-aware design of our method, we deploy it on four distinct GPU configurations, denoted S1–S4. S1 is equipped with an NVIDIA A100 (40GB), S2 with an NVIDIA A40 (48GB), S3 with an NVIDIA RTX A6000 (48GB), and S4 with an NVIDIA RTX 3090 (24GB). Additional hardware specifications, including CPU models, memory capacity, and CUDA versions, are summarized in Appendix A.

Metrics. Since our method preserves both the model architecture and acceptance criteria, it achieves acceleration without altering output quality. Therefore, we focus on efficiency metrics:

- **Speedup Ratio:** The runtime improvement relative to standard autoregressive decoding.
- **Average Acceptance Length (τ):** The mean number of tokens successfully accepted from each draft during a single draft–verify cycle.

Baselines. We use vanilla autoregressive decoding as the primary baseline (normalized to a speedup ratio of 1.0). In addition, we compare against several strong training-free speculative decoding approaches, including Lookahead Decoding (Fu et al., 2024), SPECTRA (Le et al., 2025), Prompt Lookup Decoding (PLD) (Saxena, 2023), Token Recycling (Luo et al., 2025), and training-based methods of EAGLE-1/2 (Li et al., 2024c,b). Implementation details for all baselines and our proposed method are documented in Appendix A.

5 Results

5.1 Main Results

Table 1 presents the speedup ratios of OPTISPEC and baseline methods across different models and

Model	Method	MT		Trans		Sum		QA		Math		RAG		Code		AVG
		Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.
NVIDIA A100-PCIE-40GB																
L3-8B	PLD	1.15	1.29	1.12	1.30	1.36	1.64	1.10	1.27	1.29	1.53	1.41	1.60	1.19	1.40	1.23
	Lookahead	1.38	2.00	1.34	1.88	1.28	1.98	1.38	1.95	1.50	2.01	1.28	2.02	1.39	2.03	1.36
	SPECTRA	1.44	2.00	1.35	1.87	1.40	1.99	1.37	1.90	1.45	1.94	1.49	2.10	1.42	2.01	1.42
	TokenRec.	1.93	2.58	1.85	2.48	1.94	2.57	1.85	2.45	2.06	2.72	2.18	2.89	1.96	2.69	1.97
	OPTISPEC (ours)	**2.10**	**2.91**	**1.99**	**2.77**	**2.12**	**2.93**	**1.97**	**2.72**	**2.26**	**3.10**	**2.38**	**3.25**	**2.08**	**2.94**	**2.13**
Q3-8B	PLD	1.16	1.27	1.04	1.08	1.28	1.47	1.15	1.31	1.32	1.53	1.30	1.44	1.38	1.56	1.23
	TokenRec.	1.97	2.42	1.45	1.93	1.86	2.30	1.85	2.39	2.24	2.75	1.95	2.48	2.20	2.66	1.93
	OPTISPEC (ours)	**2.07**	**2.61**	**1.76**	**2.40**	**2.03**	**2.56**	**2.00**	**2.60**	**2.34**	**2.97**	**2.06**	**2.69**	**2.27**	**2.82**	**2.07**
Q3-14B	PLD	1.15	1.26	1.03	1.08	1.23	1.42	1.10	1.25	1.30	1.51	1.21	1.32	1.47	1.63	1.21
	TokenRec.	1.94	2.41	1.34	1.93	1.76	2.30	1.79	2.39	2.25	2.80	1.76	2.36	2.19	2.69	1.86
	OPTISPEC (ours)	**2.02**	**2.68**	**1.70**	**2.42**	**1.87**	**2.60**	**1.87**	**2.53**	**2.34**	**3.05**	**1.80**	**2.57**	**2.23**	**2.90**	**1.98**
NVIDIA A40																
L3-8B	PLD	1.21	1.29	1.16	1.30	1.39	1.64	1.14	1.29	1.36	1.53	1.41	1.59	1.27	1.40	1.28
	Lookahead	1.32	2.01	1.29	1.87	1.20	2.00	1.30	1.96	1.40	2.01	1.21	2.01	1.35	2.04	1.30
	SPECTRA	1.44	2.00	1.39	1.85	1.32	1.98	1.37	1.89	1.46	1.94	1.38	2.09	1.43	1.98	1.40
	TokenRec.	1.91	2.59	1.83	2.45	1.85	2.56	1.81	2.42	2.05	2.72	2.11	2.88	1.99	2.70	1.94
	OPTISPEC (ours)	**2.08**	**2.88**	**2.02**	**2.77**	**2.04**	**2.88**	**1.99**	**2.71**	**2.25**	**3.06**	**2.32**	**3.24**	**2.11**	**2.91**	**2.12**
Q3-8B	PLD	1.18	1.27	1.03	1.08	1.23	1.46	1.16	1.32	1.35	1.53	1.23	1.43	1.42	1.58	1.23
	TokenRec.	1.79	2.40	1.31	1.93	1.64	2.29	1.71	2.38	2.04	2.73	1.67	2.46	1.98	2.64	1.73
	OPTISPEC (ours)	**1.90**	**2.62**	**1.59**	**2.39**	**1.78**	**2.56**	**1.83**	**2.60**	**2.17**	**2.96**	**1.75**	**2.64**	**2.07**	**2.82**	**1.87**
Q3-14B	PLD	1.20	1.26	1.04	1.08	1.26	1.42	1.13	1.25	1.37	1.52	1.23	1.32	1.51	1.63	1.25
	TokenRec.	1.90	2.45	1.31	1.83	1.72	2.29	1.75	2.32	2.17	2.79	1.74	2.38	2.09	2.68	1.81
	OPTISPEC (ours)	**2.00**	**2.64**	**1.64**	**2.37**	**1.89**	**2.57**	**1.84**	**2.50**	**2.32**	**3.03**	**1.83**	**2.57**	**2.21**	**2.89**	**1.96**
NVIDIA RTX A6000																
L3-8B	PLD	1.21	1.29	1.18	1.30	1.43	1.64	1.16	1.29	1.41	1.53	1.42	1.59	1.31	1.40	1.30
	Lookahead	1.29	2.00	1.26	1.85	1.18	2.00	1.29	1.96	1.37	2.01	1.21	2.03	1.34	2.03	1.28
	SPECTRA	1.42	1.99	1.37	1.84	1.30	1.98	1.38	1.90	1.43	1.92	1.41	2.11	1.43	2.01	1.39
	TokenRec.	1.89	2.59	1.84	2.46	1.84	2.56	1.81	2.43	2.04	2.72	2.11	2.92	1.93	2.68	1.92
	OPTISPEC (ours)	**2.04**	**2.87**	**2.00**	**2.75**	**2.00**	**2.86**	**1.99**	**2.72**	**2.24**	**3.05**	**2.29**	**3.24**	**2.09**	**2.91**	**2.09**
Q3-8B	PLD	1.20	1.27	1.01	1.08	1.26	1.47	1.17	1.33	1.37	1.53	1.25	1.44	1.46	1.58	1.25
	TokenRec.	1.77	2.41	1.29	1.93	1.63	2.32	1.69	2.37	2.01	2.73	1.66	2.48	1.97	2.65	1.72
	OPTISPEC (ours)	**1.87**	**2.60**	**1.60**	**2.42**	**1.77**	**2.57**	**1.83**	**2.61**	**2.14**	**2.93**	**1.75**	**2.67**	**2.05**	**2.80**	**1.86**
Q3-14B	PLD	1.23	1.26	1.04	1.08	1.29	1.42	1.15	1.25	1.41	1.52	1.23	1.32	1.54	1.63	1.27
	TokenRec.	1.94	2.42	1.36	1.85	1.73	2.29	1.77	2.31	2.21	2.78	1.70	2.35	2.14	2.68	1.84
	OPTISPEC (ours)	**2.04**	**2.65**	**1.64**	**2.35**	**1.87**	**2.57**	**1.88**	**2.52**	**2.35**	**3.02**	**1.77**	**2.55**	**2.23**	**2.88**	**1.97**
NVIDIA GeForce RTX 3090																
L3-8B	PLD	1.32	1.29	1.25	1.30	1.46	1.64	1.23	1.29	1.46	1.53	1.33	1.60	1.28	1.40	1.33
	Lookahead	1.15	2.00	1.15	1.90	1.05	1.99	1.14	1.98	1.21	2.02	1.07	2.02	1.11	2.01	1.13
	SPECTRA	1.25	1.96	1.07	1.79	1.24	2.01	1.34	1.90	1.41	1.94	1.35	2.07	1.32	1.98	1.28
	TokenRec.	1.51	2.57	1.47	2.47	1.49	2.59	1.44	2.42	1.63	2.73	1.68	2.89	1.47	**2.68**	1.53
	OPTISPEC (ours)	**1.96**	**2.58**	**1.92**	**2.48**	**1.91**	**2.59**	**1.91**	**2.46**	**2.15**	**2.74**	**2.25**	**3.00**	**1.85**	2.59	**1.99**
Q3-8B	PLD	1.21	1.27	1.06	1.08	1.30	1.47	1.19	1.33	1.37	1.53	1.29	1.43	1.45	1.58	1.27
	TokenRec.	1.63	**2.40**	1.23	1.94	1.47	**2.29**	1.55	**2.38**	1.87	**2.74**	1.49	**2.44**	1.80	**2.63**	1.58
	OPTISPEC (ours)	**1.78**	2.33	**1.46**	**2.09**	**1.67**	2.22	**1.71**	2.37	**2.00**	2.63	**1.70**	2.37	**1.89**	2.46	**1.75**

Table 1: Spec-Bench results across tasks of Spec-Bnch and testbeds (S1–S4). For each model and method we report speedup (Spd., vs. autoregressive=1.0) and average acceptance length τ; **AVG** is the macro-average over tasks. L3-xB = Llama-3-Instruct (xB), Q3-xB = Qwen-3 (xB).

hardware on Spec-Bench tasks. Across all environment settings, OPTISPEC consistently achieves the strongest speedups, improving runtime by 10%–30% compared with prior training-free methods. Notably, our method maintains a stable 2× speedup even on resource-constrained GPUs such as the RTX 3090, where other baselines show substantial degradation. This demonstrates the effectiveness of our environment-aware strategy: calibrating the draft length to the device preserves the τ/s trade-off and keeps the verification process efficient across heterogeneous GPU platforms. In the per-task breakdown, OPTISPEC ranks first across all evaluated tasks, confirming its effectiveness and

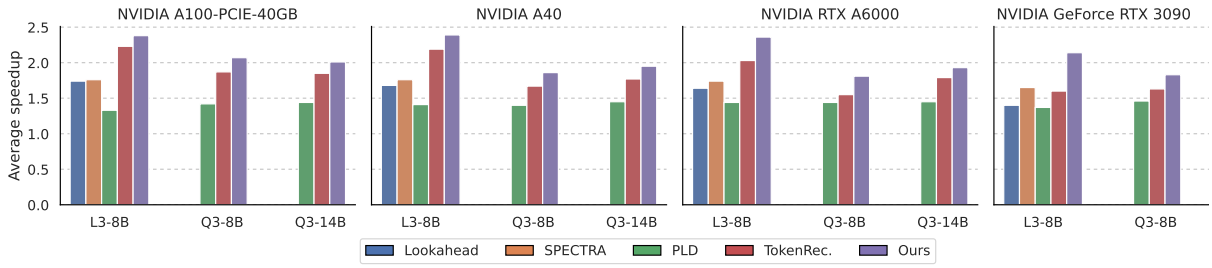

Figure 4: Average speedup across languages of Multi-SpecBench for each testbed (S1–S4).

robustness in diverse decoding scenarios.

Regarding the average acceptance length (τ)—the mean number of tokens accepted per verification step—our method consistently achieves higher τ than the baselines, reaching up to $3.1\times$ improvement. A larger τ means more tokens pass verification per step, reducing the total number of verification rounds to produce the same output length. This gain stems from our dynamic speculation tree with a tree expansion mechanism that broadens high-value first-level candidates and filters low-confidence branches using adaptive thresholds, while rebuilding the draft tree at each step based on confidence scores.

Multilingual robustness. Figure 4 presents the average performance of Multi-SpecBench across all languages on four hardware configurations (detailed results are provided in Appendix D). OPTISPEC achieves the best cross-language averages: for Llama3-8B, the average speedups reach $2.38\times$ (A100), $2.39\times$ (A40), $2.36\times$ (RTX A6000), and $2.14\times$ (RTX 3090), consistently outperforming all baselines under the same setups. We also observe a clear trend on lower-resource GPUs, such as the RTX A6000 and RTX 3090, where the performance gap between OPTISPEC and the baselines widens. We attribute this to previous methods being primarily tuned for optimal draft lengths on high-end GPUs, whereas our approach adapts effectively to different hardware. Results on Qwen3-8B and Qwen3-14B further confirm that OPTISPEC consistently delivers the best performance across models.

5.2 Analysis

Ablation Study. We analyze the contribution of each OPTISPEC component using LLaMA3-8B-Instruct on an RTX 3090 with Spec-Bench (Table 2). Fixing the draft length to match Token Recycling (Luo et al., 2025) (*w/o device-aware calibration*) reduces speedups to $1.61\times/1.52\times$ (MT-Bench/Code), showing that fixed draft lengths fail to generalize across hardware. Removing the *score-based tree-node estimation* (replacing it with the static template of the baseline (Luo et al., 2025)) further lowers performance to $1.48\times/1.46\times$, indicating that confidence-weighted branching yields stronger draft candidates. Disabling the *tree expansion mechanism* — i.e., removing our newly introduced expansion strategy that leverages the token preceding the root and applies threshold-based filtering — reduces speedups to $1.56\times/1.51\times$ and decreases τ (2.07/2.16), showing that without the enhanced expansion, acceptance and throughput drop. Despite these degradations, all variants are still comparable to existing baselines, highlighting the complementary strengths of each component.

	MT Bench		Code	
Method	Spd.	τ	Spd.	τ
SPECTRA	1.25	1.96	1.32	1.98
TokenRec.	1.51	2.57	1.47	2.68
OPTISPEC (ours)	**1.96**	**2.58**	**1.85**	2.59
- *w/o device-aware calibration*	1.61	2.76	1.52	**2.79**
- *w/o score estimation*	1.48	2.62	1.46	2.75
- *w/o tree expansion mechanism*	1.66	2.20	1.63	2.29

Table 2: Ablation study of OPTISPEC on RTX 3090 (Llama-3-8B-Instruct, Spec-Bench) for multi-turn conversation and code generation tasks.

Comparison to training-based methods. We further compare OPTISPEC with state-of-the-art training-based methods: EAGLE-1 (Li et al., 2024c) and EAGLE-2 (Li et al., 2024b). EAGLE-1 trains a small drafter using the target model's hidden states, while EAGLE-2 extends it with dynamic tree construction. Although both deliver strong speedups, they require data collection, training, and maintenance of auxiliary models. As shown in Table 3, OPTISPEC matches EAGLE's performance on English and outperforms it on Japanese, where the trained drafter struggles with cross-lingual gen-

eralization. This demonstrates the strength of our training-free, plug-and-play framework, which achieves robust, multilingual efficiency without retraining or additional data.

Method	EN		JA		DE	
	Spd.	τ	Spd.	τ	Spd.	τ
EAGLE-1	1.81	3.48	1.17	2.32	1.13	2.19
EAGLE-2	**2.18**	**4.21**	1.18	2.34	1.14	2.21
OPTISPEC (ours)	2.13	2.94	**2.45**	**3.40**	**2.21**	**3.08**

Table 3: Comparison with training-based methods on Multi-SpecBench for English, Japanese, and German.

Optimal values of draft tokens. Figure 5 shows the device-specific optima $g^\star$ selected by our calibration, confirming that the optimal draft length is *not universal* but strongly tied to verification cost. On the slower, memory-constrained RTX 3090, the forward pass scales poorly with sequence length, so a smaller draft size ($g^\star \approx 50$–60 tokens) maximizes τ/s by reducing verification overhead. In contrast, the data-center A100 can sustain higher parallelism and throughput with longer inputs, pushing the optimum to about $g^\star \approx 110$ tokens. This clear separation across the four environments underscores the necessity of environment-aware calibration: acceptance gains from larger drafts saturate quickly, and without tuning, extra draft tokens can inflate s enough to reduce overall speed despite similar τ.

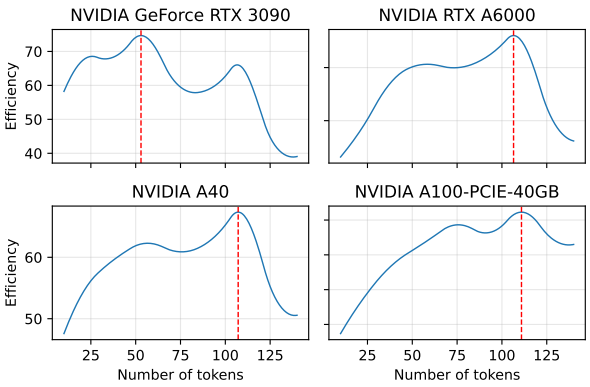

Figure 5: Calibrated optimal draft size $g^\star$ by enviroment settings (S1–S4).

Other Analysis. Detailed speedup and acceptance length results for all languages appear in Appendix D. Additional evaluations under different sampling temperatures are in Appendix E, multi-GPU performance in Appendix F, and calibration size/time analysis in Appendix G.2.

6 Related Works

Efficient inference is essential for both real-time and low-resource applications, motivating numerous methods to reduce decoding latency (Ma et al., 2023; Gu et al., 2024; Liu et al., 2024b). Among them, *speculative decoding* (Chen et al., 2023; Xia et al., 2024; Leviathan et al., 2023) stands out for its lossless nature, ensuring identical outputs to the target model. It follows a draft-and-verify paradigm (Le et al., 2025), where a drafter proposes multiple candidate tokens and the target model verifies them in parallel, enabling multi-token generation per step. Depending on how the drafter is obtained, methods fall into two categories: **training-based** and **training-free**. Training-based methods such as **Medusa** (Cai et al., 2024) and **EAGLE-1/2** (Li et al., 2024c,b) train auxiliary drafters to mimic target model predictions, while **FR-Spec** (Zhao et al., 2025) and **HASS** (Zhang et al., 2025b) further optimize architecture and training alignment. Although effective, these approaches incur substantial overhead in data collection, fine-tuning, and cross-device adaptation.

Training-free methods eliminate retraining by exploiting existing model predictions or external data. Retrieval-based approaches like **REST** (He et al., 2024) reuse datastore continuations, while **Token Recycling** (Luo et al., 2025) dynamically reuses previously discarded tokens. Jacobi-style methods such as **Lookahead** (Fu et al., 2024) and **SPECTRA** (Le et al., 2025) generate candidates directly from model probabilities. Despite their practicality, these methods often employ fixed draft templates and overlook device-specific verification costs, limiting efficiency on different hardware.

7 Conclusions

We introduced OPTISPEC, a training-free and device-calibrated speculative decoding framework that adaptively optimizes draft size and token selection for efficient large language model inference. Through hardware calibration, confidence-guided tree construction, and enhanced expansion strategies, OPTISPEC achieves robust, plug-and-play acceleration without additional training. Experiments on Multi-SpecBench demonstrate consistent improvements across languages, tasks, and hardware, delivering up to 2.6× speedup over existing training-free methods.

8 Limitations

Although OptiSpec provides practical and training-free acceleration for large language models, several limitations remain.

Hardware Stability. Our device-aware calibration assumes relatively stable hardware performance and may be less reliable in highly dynamic or resource-shared environments (e.g., multi-tenant GPUs or throttled inference servers). While the calibration is lightweight, sudden changes in GPU memory bandwidth or scheduling can reduce the accuracy of the estimated draft size $g^\star$ and lead to suboptimal speedups.

Language Coverage. Our evaluation primarily covers languages with moderately analytic morphology (e.g., English, German, French, Spanish, Chinese, Japanese, Vietnamese) and has not yet been extended to languages with richer morphology or complex orthography, such as Arabic or highly agglutinative languages. These languages could yield lower acceptance rates due to more unpredictable token boundaries, and additional adaptation might be required for robust performance.

Logit Access Requirement. Although OptiSpec does not require retraining, it still depends on access to the model's logits during decoding. This assumption limits applicability in strictly black-box deployment settings where only next-token sampling is available, such as certain closed APIs.

These limitations point to important future directions, including dynamic re-calibration under fluctuating compute availability, systematic evaluation on morphologically rich and low-resource languages, and adaptation for closed API models.

References

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637, Online. Association for Computational Linguistics.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen Technical Report.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 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.

Oralie Cattan, Christophe Servan, and Sophie Rosset. 2021. On the Usability of Transformers-based models for a French Question-Answering task. In *Recent Advances in Natural Language Processing (RANLP)*, Varna, Bulgaria.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Cohere. 2023. wikipedia-2023-11-embed-multilingual-v3. Dataset on Hugging Face. Accessed 2025-07-31.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, Singapore. Association for Computational Linguistics.

Michael Dinzinger, Laura Caspari, Kanishka Ghosh Dastidar, Jelena Mitrović, and Michael Granitzer. 2025. Webfaq: A multilingual collection of natural qa datasets for dense retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, page 3802–3811, New York, NY, USA. Association for Computing Machinery.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. 2024. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642, Bangkok, Thailand. Association for Computational Linguistics.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 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.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-Based Speculative Decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. 2025. SAM Decoding: Speculative Decoding via Suffix Automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12187–12204, Vienna, Austria. Association for Computational Linguistics.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B.

Wouter Kool, Herke van Hoof, and Max Welling. 2020. Ancestral Gumbel-Top-k Sampling for Sampling Without Replacement. *Journal of Machine Learning Research*, 21(47):1–36.

Kentaro Kurihara, Daisuke Kawahara, and Tomohide Shibata. 2022. JGLUE: Japanese general language understanding evaluation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2957–2966, Marseille, France. European Language Resources Association.

Nguyen-Khang Le, Truong Dinh Do, and Le-Minh Nguyen. 2025. SPECTRA: Faster large language model inference with optimized internal and external speculation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14015–14034, Vienna, Austria. Association for Computational Linguistics.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 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.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Wen-tau Yih, and Xi Victoria Lin. 2024a. Nearest Neighbor Speculative Decoding for LLM Generation and Attribution. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432, Miami, Florida, USA. Association for Computational Linguistics.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024c. Eagle: speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. 2024a. Kangaroo: Lossless Self-Speculative Decoding for Accelerating LLMs via Double Early Exiting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Tianyu Liu, Qitan Lv, Hao Li, Xing Gao, and Xiao Sun. 2025. LogitSpec: Accelerating Retrieval-based Speculative Decoding via Next Next Token Speculation. _eprint: 2507.01449.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024b. LLM-QAT: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 467–484, Bangkok, Thailand. Association for Computational Linguistics.

Hoang Long. 2025. Vi-gsm8k: Vietnamese version of the gsm8k math word problem dataset. `https://huggingface.co/datasets/longhoang06/Vi-GSM8K`. Accessed: 2025-07-31.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. 2025. Turning trash into treasure: Accelerating inference of large language models with token recycling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6816–6831, Vienna, Austria. Association for Computational Linguistics.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 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, pages 932–949, New York, NY, USA. Association for Computing Machinery. Event-place: La Jolla, CA, USA.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. GPT-4 Technical Report.

Jie Ou, Yueming Chen, and Prof. Tian. 2024. Lossless Acceleration of Large Language Model via Adaptive N-gram Parallel Decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 10–22, Mexico City, Mexico. Association for Computational Linguistics.

Qiwei Peng, Yekun Chai, and Xuhong Li. 2024. HumanEval-XL: A multilingual code generation benchmark for cross-lingual natural language generalization. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 8383–8394, Torino, Italia. ELRA and ICCL.

Xiaoye Qu, Yafu Li, Zhaochen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, Peng Li, Wei Wei, Jing Shao, Chaochao Lu, Yue Zhang, Xian-Sheng Hua, Bowen Zhou, and Yu Cheng. 2025. A Survey of Efficient Reasoning for Large Reasoning Models: Language, Multimodality, and Beyond. _eprint: 2503.21614.

Miyu Sato, Shiho Takano, Teruno Kajiura, and Kimio Kuramitsu. 2024. Does the llm demonstrate cross-lingual knowledge transfer by additional japanese training? In *Proceedings of the Thirtieth Annual Meeting of the Association for Natural Language Processing*.

Apoorv Saxena. 2023. Prompt lookup decoding.

Rainer Storn and Kenneth Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2023. SpecTr: fast speculative decoding via optimal transport. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc. Event-place: New Orleans, LA, USA.

Lightblue Tech. 2023. Multilingual mt bench. https://github.com/lightblue-tech/multilingual-mt-bench. Accessed: 2025-07-31.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent abilities of large language models. *Transactions on Machine Learning Research*. Survey Certification.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7655–7671, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Inference with Reference: Lossless Acceleration of Large Language Models. _eprint: 2304.04487.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2024. Predictive Pipelined Decoding: A Compute-Latency Trade-off for Exact LLM Decoding. *Transactions on Machine Learning Research*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2024. Draft& Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11263–11282, Bangkok, Thailand. Association for Computational Linguistics.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, Yu Fu, Xingtai Lv, Yuchen Zhang, Sihang Zeng, Shang Qu, Haozhan Li, Shijie Wang, Yuru Wang, Xinwei Long, and 20 others. 2025a. A Survey of Reinforcement Learning for Large Reasoning Models. _eprint: 2509.08827.

Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. 2025b. Learning harmonized representations for speculative sampling. In *The Thirteenth International Conference on Learning Representations*.

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Sun Ao, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jie Zhou, Hao Zhou, Jianyong Wang, Maosong Sun, and Zhiyuan Liu. 2025. FR-Spec: Accelerating Large-Vocabulary Language Models via Frequency-Ranked Speculative Sampling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3909–3921, Vienna, Austria. Association for Computational Linguistics.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. In *The Twelfth International Conference on Learning Representations*.

A Implementation and Reproducibility Details

Frameworks. All experiments use PyTorch 2.8.0 and Transformers 4.56.2, with static KV cache allocation and FP16 precision unless otherwise noted.

Models. We evaluate **Llama-3-8B-Instruct** (Dubey et al., 2024), **Qwen-3-8B**, and **Qwen-3-14B** (Yang et al., 2025). All checkpoints are taken from official repositories without additional training or modifications.

Decoding setup. Batch size is 1 for all runs. For the generation setting, we cap the maximum new tokens at 1024. Seeds are fixed to 0. Our method keeps verifier logits identical to standard decoding, ensuring lossless outputs under the same sampling policy (e.g., greedy).

Baselines. We re-run each baseline with authors' public code and recommended defaults: Lookahead (Fu et al., 2024), SPECTRA (Le et al., 2025), Token Recycling (Luo et al., 2025), and EAGLE-2 (Li et al., 2024b).

OptiSpec Hyperparameters. We set generous upper bounds for the draft tree and rely on pruning to select effective candidates. The maximum tree depth is set at $d = 10$, and at each level we retain the top-$k = 10$ branches ranked by confidence scores (Algorithm 1). A confidence threshold r is swept over $\{0.01, 0.025, \mathbf{0.05}, 0.1, 0.2\}$[1]. These settings ensure that pruning does not prematurely discard promising nodes. The total number of drafted tokens g is not fixed; instead, it is determined by our device-aware calibration on 5 samples from Ultrachat (Ding et al., 2023), which selects $g^\star$ to maximize τ/s for each hardware configuration. For an analysis of calibration sample size, see Appendix G.2. For the n-gram storage mechanism, we adopt the one-to-one strategy (Luo et al., 2025), maintaining two $[V, k]$ matrices: an adjacency matrix storing next-token candidates for each vocabulary token and a confidence matrix with the corresponding softmax probabilities from the verifier's logits; both are updated at every decoding step and used to compute node confidence scores for draft tree expansion.

Testbed Specification. Table 4 lists the four environments (S1–S4) used throughout the paper. All experiments ran on Linux with the vendor GPU drivers matching the CUDA versions shown in the table.

B Training-free N-gram Strategies

Several prior works employ either a trained drafter model (Chen et al., 2023; Leviathan et al., 2023; Miao et al., 2024; Sun et al., 2023; Zhou et al., 2024; Cai et al., 2024) or modify the original LLM in a specialized manner (self-speculative decoding) (Elhoushi et al., 2024; Liu et al., 2024a; Yang et al., 2024; Zhang et al., 2024; Li et al., 2024b) to predict multiple tokens in advance. While effective, these approaches require costly training and may alter the inherent capabilities of the original LLM.

An alternative line of research focuses on predicting subsequent tokens without any additional training. This paradigm preserves the original model and enables off-the-shelf deployment, avoiding the need for auxiliary models or fine-tuning. In these methods, draft tokens are obtained directly from the LLM's predictions (Fu et al., 2024; Ou et al., 2024) or from external information sources (Yang et al., 2023; He et al., 2024; Li et al., 2024a). Typically, such approaches maintain an n-gram store, equipped with mechanisms for updating stored n-grams and strategies for retrieving them to gener-

[1] Bold values indicate the best result on the calibration data

ID	GPU	VRAM	CPU	RAM	CUDA	OS
S1	NVIDIA A100 (PCIe)	40 GB	52 × Intel(R) Xeon(R) Gold 5320 CPU @ 2.20 GHz	503 GB	12.8	Ubuntu 20.04
S2	NVIDIA A40	48 GB	52 × Intel(R) Xeon(R) Gold 5320 CPU @ 2.20 GHz	503 GB	12.8	Ubuntu 20.04
S3	NVIDIA RTX A6000	48 GB	32 × Intel(R) Xeon(R) Gold 6346 CPU @ 3.10 GHz	504 GB	12.3	Ubuntu 20.04
S4	NVIDIA GeForce RTX 3090	24 GB	8 × Intel(R) Xeon(R) W-2223 CPU @ 3.60 GHz	126 GB	12.0	Ubuntu 18.04

Table 4: Hardware and software configuration of the four testbeds (S1–S4).

ate draft candidates. Below, we describe common training-free n-gram strategies.

One-to-Many. The n-gram store maintains sequences of length n, indexed by their first token. Given the last token of the input sequence, we query the index and retrieve all matching n-grams. For each retrieved n-gram, the remaining $n-1$ tokens are used as a draft sequence. This procedure yields m draft sequences, each of length $n-1$. This strategy is employed in Fu et al. (2024); He et al. (2024); Le et al. (2025).

Many-to-One. Here, the n-gram store also maintains sequences of length n. Given the last $n-1$ tokens of the input, we search for n-grams in the store that begin with these tokens. The final token of each matching n-gram is then used as a draft candidate. This step is repeated until reaching a predefined draft size G, resulting in a single draft sequence of length G. This strategy is employed in Ou et al. (2024); Le et al. (2025).

One-to-One. In this strategy, the store maintains 2-grams, indexed by their first token. Given the last token of the input, we query the store for all 2-grams beginning with that token. The second token of each retrieved 2-gram is taken as a drafting candidate. Since multiple 2-grams may share the same prefix, the draft tree expands into multiple branches at each step. This expansion is repeated until reaching depth G, corresponding to a draft size of G. This strategy is used in Luo et al. (2025).

C Details of Multi-SpecBench Construction

We extend SpecBench into a multilingual benchmark, **Multi-SpecBench**, by selecting 80 examples for each task–language pair across six languages: **German, French, Spanish, Chinese, Japanese, and Vietnamese**. Given seven tasks, this setup yields a total of 560 samples per language. The data sources for each task are as follows:

- **Code Generation:** Programming tasks from `HumanEval-XL` (Peng et al., 2024), except for Japanese, where we use `jhumaneval` (Sato et al., 2024).
- **Multi-turn Conversation:** Interactive dialogue prompts taken from Multilingual MT-Bench (Tech, 2023).
- **Question Answering (QA):** Factual queries selected from `webfaq` (Dinzinger et al., 2025).
- **Summarization:** Passages from multilingual Wikipedia, sourced from `wikipedia-embed-multilingual-v3` (Cohere, 2023), paired with prompts requesting concise summaries.
- **Translation:** English text segments drawn from `wikitext` (Merity et al., 2017), accompanied by prompts to translate them into each target language.
- **Retrieval-Augmented Generation (RAG):** Question–context pairs primarily from `xquad` (Artetxe et al., 2020); for French and Japanese, we substitute with `squad_fr` (Cattan et al., 2021) and `JSQuAD` (Kurihara et al., 2022), respectively.
- **Math Reasoning:** Reasoning problems based on `mgsm` (Wei et al., 2022), with Vietnamese items adapted from `vi_gsm8k` (Long, 2025).

All inputs are standardized to maintain consistent phrasing, style, and expected output format, following the design principles of Spec-Bench (Xia et al., 2024).

Model	Method	DE		ES		FR		JA		VI		ZH		AVG
		Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.
NVIDIA A100-PCIE-40GB														
L3-8B	PLD	1.22	1.36	1.15	1.30	1.24	1.39	1.57	1.65	1.35	1.51	1.44	1.45	1.33
	Lookahead	1.58	2.46	1.61	2.49	1.63	2.51	1.91	2.83	1.85	2.74	1.88	2.78	1.74
	SPECTRA	1.57	2.33	1.61	2.37	1.65	2.44	1.93	2.73	1.87	2.62	1.93	2.67	1.76
	TokenRec.	2.07	2.69	2.17	2.86	2.13	2.81	2.29	2.95	2.44	3.12	2.29	2.95	2.23
	OPTISPEC (ours)	**2.21**	**3.08**	**2.28**	**3.23**	**2.28**	**3.22**	**2.45**	**3.41**	**2.61**	**3.62**	**2.46**	**3.39**	**2.38**
Q3-8B	PLD	1.34	1.37	1.38	1.37	1.35	1.37	1.66	1.55	1.40	1.45	1.40	1.38	1.42
	TokenRec.	1.78	2.25	1.83	2.32	1.85	2.31	1.92	2.49	1.97	2.52	1.89	2.41	1.87
	OPTISPEC (ours)	**1.97**	**2.55**	**2.03**	**2.62**	**2.05**	**2.62**	**2.17**	**2.85**	**2.22**	**2.89**	**2.01**	**2.60**	**2.07**
Q3-14B	PLD	1.40	1.34	1.39	1.33	1.40	1.34	1.64	1.48	1.42	1.44	1.40	1.37	1.44
	TokenRec.	1.74	2.20	1.76	2.27	1.81	2.30	1.91	2.51	1.95	2.51	1.92	2.45	1.85
	OPTISPEC (ours)	**1.89**	**2.48**	**1.92**	**2.60**	**2.00**	**2.62**	**2.10**	**2.83**	**2.15**	**2.87**	**2.01**	**2.63**	**2.01**
NVIDIA A40														
L3-8B	PLD	1.29	1.36	1.22	1.31	1.31	1.39	1.70	1.66	1.43	1.44	1.52	1.45	1.41
	Lookahead	1.52	2.45	1.53	2.47	1.61	2.58	1.83	2.84	1.77	2.75	1.80	2.77	1.68
	SPECTRA	1.54	2.32	1.59	2.35	1.64	2.41	1.92	2.71	1.91	2.76	1.93	2.80	1.76
	TokenRec.	1.97	2.79	2.08	2.94	2.10	3.00	2.09	2.80	2.54	3.38	2.35	3.17	2.19
	OPTISPEC (ours)	**2.22**	**3.07**	**2.27**	**3.19**	**2.30**	**3.20**	**2.46**	**3.40**	**2.63**	**3.59**	**2.46**	**3.37**	**2.39**
Q3-8B	PLD	1.31	1.37	1.35	1.37	1.34	1.37	1.63	1.55	1.38	1.46	1.38	1.38	1.40
	TokenRec.	1.59	2.27	1.62	2.31	1.65	2.32	1.72	2.50	1.76	2.52	1.69	2.41	1.67
	OPTISPEC (ours)	**1.77**	**2.55**	**1.80**	**2.61**	**1.82**	**2.59**	**1.95**	**2.84**	**1.99**	**2.87**	**1.81**	**2.58**	**1.86**
Q3-14B	PLD	1.41	1.35	1.40	1.33	1.41	1.34	1.66	1.48	1.42	1.43	1.41	1.37	1.45
	TokenRec.	1.66	2.19	1.70	2.28	1.73	2.29	1.83	2.51	1.88	2.52	1.84	2.44	1.77
	OPTISPEC (ours)	**1.82**	**2.46**	**1.88**	**2.60**	**1.92**	**2.59**	**2.04**	**2.84**	**2.09**	**2.86**	**1.95**	**2.62**	**1.95**
NVIDIA RTX A6000														
L3-8B	PLD	1.33	1.36	1.25	1.31	1.36	1.39	1.72	1.65	1.46	1.44	1.56	1.45	1.44
	Lookahead	1.50	2.42	1.48	2.43	1.54	2.50	1.79	2.82	1.76	2.76	1.79	2.78	1.64
	SPECTRA	1.56	2.32	1.59	2.33	1.65	2.43	1.92	2.76	1.86	2.64	1.90	2.71	1.74
	TokenRec.	1.89	2.69	1.94	2.85	1.90	2.80	2.10	2.97	2.25	3.09	2.07	2.94	2.03
	OPTISPEC (ours)	**2.20**	**3.08**	**2.26**	**3.19**	**2.26**	**3.19**	**2.41**	**3.36**	**2.60**	**3.59**	**2.43**	**3.34**	**2.36**
Q3-8B	PLD	1.34	1.37	1.40	1.37	1.39	1.37	1.65	1.55	1.43	1.46	1.44	1.38	1.44
	TokenRec.	1.48	2.27	1.51	2.31	1.54	2.32	1.59	2.48	1.63	2.50	1.57	2.40	1.55
	OPTISPEC (ours)	**1.73**	**2.53**	**1.77**	**2.62**	**1.79**	**2.59**	**1.90**	**2.82**	**1.93**	**2.87**	**1.75**	**2.57**	**1.81**
Q3-14B	PLD	1.41	1.35	1.39	1.33	1.42	1.34	1.65	1.48	1.42	1.43	1.42	1.36	1.45
	TokenRec.	1.67	2.21	1.69	2.29	1.75	2.30	1.85	2.52	1.89	2.54	1.87	2.47	1.79
	OPTISPEC (ours)	**1.82**	**2.48**	**1.85**	**2.59**	**1.92**	**2.59**	**2.02**	**2.82**	**2.06**	**2.85**	**1.93**	**2.61**	**1.93**
NVIDIA GeForce RTX 3090														
L3-8B	PLD	1.27	1.36	1.20	1.31	1.29	1.39	1.62	1.65	1.40	1.46	1.47	1.44	1.37
	Lookahead	1.29	2.46	1.28	2.46	1.35	2.57	1.52	2.84	1.45	2.73	1.49	2.77	1.40
	SPECTRA	1.46	2.35	1.51	2.34	1.57	2.45	1.82	2.77	1.75	2.63	1.77	2.67	1.65
	TokenRec.	1.47	2.68	1.54	2.83	1.54	2.80	1.65	2.96	1.73	3.08	1.65	2.95	1.60
	OPTISPEC (ours)	**1.99**	**2.79**	**2.05**	**2.90**	**2.05**	**2.90**	**2.19**	**3.06**	**2.36**	**3.25**	**2.21**	**3.05**	**2.14**
Q3-8B	PLD	1.39	1.37	1.42	1.37	1.41	1.37	1.68	1.55	1.45	1.46	1.44	1.38	1.46
	TokenRec.	1.55	**2.27**	1.60	**2.34**	1.62	**2.32**	1.65	2.49	1.71	**2.51**	1.64	**2.40**	1.63
	OPTISPEC (ours)	**1.74**	2.23	**1.79**	2.31	**1.83**	2.30	**1.92**	**2.49**	**1.93**	2.49	**1.76**	2.25	**1.83**

Table 5: Full Multi-SpecBench results across devices (A100, A40, RTX A6000, RTX 3090). We report speedup (Spd.) and average acceptance length τ for DE/ES/FR/JA/VI/ZH and their average (**AVG**).

D Detailed Multilingual Benchmark Results

Table 5 reports the complete results for speedup and acceptance length across all tasks in Spec-Bench,

Model	Method	MT Spd.	MT τ	Trans Spd.	Trans τ	Sum Spd.	Sum τ	QA Spd.	QA τ	Math Spd.	Math τ	RAG Spd.	RAG τ	Code Spd.	Code τ	AVG Spd.
Temperature = 0.0																
L3-8B	Lookahead	1.38	2.00	1.34	1.88	1.28	1.98	1.38	1.95	1.50	2.01	1.28	2.02	1.39	2.03	1.36
	SPECTRA	1.44	2.00	1.35	1.87	1.40	1.99	1.37	1.90	1.45	1.94	1.49	2.10	1.42	2.01	1.42
	TokenRec.	1.93	2.58	1.85	2.48	1.94	2.57	1.85	2.45	2.06	2.72	2.18	2.89	1.96	2.69	1.97
	Ours	2.10	2.91	1.99	2.77	2.12	2.93	1.97	2.72	2.26	3.10	2.38	3.25	2.08	2.94	2.13
Temperature = 1.0																
L3-8B	Lookahead	1.38	1.69	1.39	1.93	1.29	1.96	1.42	1.98	1.50	2.01	1.34	2.04	1.42	2.00	1.39
	SPECTRA	1.40	1.59	1.35	1.81	1.42	1.97	1.39	1.91	1.46	1.95	1.51	2.09	1.41	1.99	1.42
	TokenRec.	1.81	2.36	1.76	2.29	1.78	2.30	1.71	2.21	1.93	2.45	1.69	2.15	1.78	2.37	1.78
	Ours	1.86	2.57	1.83	2.50	1.86	2.59	1.83	2.53	1.99	2.69	1.74	2.36	1.84	2.64	1.85

Table 6: Effect of sampling temperature on Spec-Bench (Llama-3-8B-Instruct). We report speedup (Spd., vs. autoregressive=1.0) and average acceptance length τ at $T = 0.0$ (greedy) and $T = 1.0$.

Model	Method	MT TP	MT Spd.	MT τ	QA TP	QA Spd.	QA τ	Math TP	Math Spd.	Math τ	Code TP	Code Spd.	Code τ	AVG Spd.
1 × NVIDIA GeForce RTX 3090														
L3-8B	Autogressive	38.26	1.00	1.00	39.09	1.00	1.00	39.10	1.00	1.00	41.55	1.00	1.00	1.00
	TokenRec.	57.75	1.51	2.57	56.19	1.44	2.42	63.63	1.63	2.73	61.06	1.47	2.68	1.51
	Ours	74.59	1.95	2.58	74.03	1.89	2.44	83.69	2.14	2.73	76.91	1.85	2.60	1.96
2 × NVIDIA GeForce RTX 3090														
L3-8B	Autogressive	32.46	1.00	1.00	32.54	1.00	1.00	32.47	1.00	1.00	32.48	1.00	1.00	1.00
	TokenRec.	50.19	1.55	2.61	47.80	1.47	2.41	54.53	1.68	2.73	52.65	1.62	2.69	1.58
	Ours	56.95	1.75	2.58	54.59	1.68	2.45	61.25	1.89	2.75	57.57	1.77	2.61	1.77

Table 7: Distributed inference on two NVIDIA GeForce RTX 3090 GPUs (tensor parallelism) versus a single RTX 3090 on Spec-Bench with Llama-3-8B-Instruct. We report throughput (TP, tokens/s), speedup (Spd.), and average acceptance length (τ) per task and the average (**AVG**).

as well as per-language performance for the multilingual Multi-SpecBench introduced in Section C. Across all languages and tasks, OPTISPEC consistently achieves higher throughput than training-free methods, as evidenced by improvements in both speedup and acceptance length, reaching up to 2.63× speedup. These results further demonstrate the robustness of OPTISPEC to diverse tasks, languages, and hardware configurations.

E Acceleration in Sampling Decoding

We compare the performance of our method under greedy decoding (temperature $= 0$) and sampling decoding (temperature $= 1.0$) on LLaMA-3-8B using Spec-Bench, as shown in Table 6. A slight performance degradation is observed at higher sampling temperatures, likely due to the lower acceptance rate during the sampling-based verification phase, consistent with prior findings (Luo et al., 2025; Le et al., 2025). Despite this, the speedup ratio of OPTISPEC remains consistently around 2.0×, demonstrating sustained acceleration and confirming the robustness and broad applicability of our approach across different decoding strategies.

F Multi-GPU / Distributed Setting

A critical consideration for practical deployment is how OPTISPEC scales when models are distributed across multiple GPUs—a common requirement for large LLMs that exceed the memory capacity of a single device. To evaluate this, we compare OPTISPEC running on a single RTX 3090 with a distributed setup spanning two RTX 3090 GPUs using tensor parallelism. Table 7 reports results on Spec-Bench with **LLaMA-3-8B-Instruct**. The distributed configuration shows slightly lower speedup than the single-GPU setup, dropping from **1.96**× on a single GPU to **1.77**× with two GPUs. We attribute this degradation to potential inter-GPU communication overhead and less efficient cache sharing, which may increase verification latency when tokens are synchronized across devices. Despite this, OPTISPEC still consistently outperforms baseline training-free methods under distributed

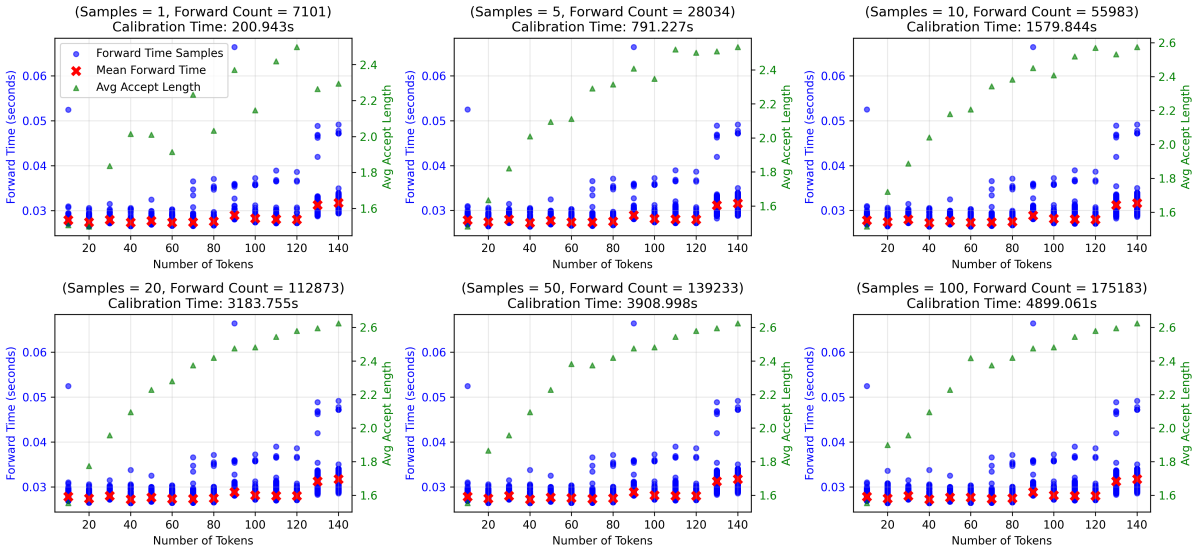

Figure 6: Forward time, average accept length, and calibration time when calibrating with different numbers of samples.

inference, demonstrating that its core acceleration benefits remain effective even when scaling beyond a single GPU.

G Calibration Details

G.1 Calibration and Optimization Procedure

To determine the optimal draft size $g^\star$, we employ an empirical calibration procedure that estimates the throughput function $f_{\text{speed}}(g) = f_{\text{accept}}(g)/f_{\text{time}}(g)$ using regression models fitted from a small number of warm-up runs.

Data collection. For each calibration run with draft size g_i, we record the average number of accepted tokens τ_i and the forward time s_i. This yields two datasets:

$$S = \{(g_1, s_1), (g_2, s_2), \ldots, (g_m, s_m)\},$$

$$T = \{(g_1, \tau_1), (g_2, \tau_2), \ldots, (g_m, \tau_m)\}.$$

Regression modeling. We model the dependency between draft size and both performance metrics via lightweight regression functions:

$$f_{\text{time}}(g) \approx s, \quad f_{\text{accept}}(g) \approx \tau.$$

In implementation, the forward time function $f_{\text{time}}(g)$ is estimated using a **B-spline regression model** (via the `SplineTransformer` and `LinearRegression` from `scikit-learn`), allowing smooth interpolation over irregular measurements. The acceptance length function $f_{\text{accept}}(g)$ is modeled using a **third-degree polynomial regression** (via `PolynomialFeatures` + `LinearRegression`), which empirically provides a good trade-off between flexibility and stability.

Throughput maximization. The estimated throughput function is defined as:

$$f_{\text{speed}}(g) = \frac{f_{\text{accept}}(g)}{f_{\text{time}}(g)}.$$

The optimal draft size $g^\star$ is then obtained by solving:

$$g^\star = \arg\max_{g} \; f_{\text{speed}}(g).$$

Because the throughput landscape may be non-convex, we employ the **Differential Evolution (DE)** algorithm (Storn and Price, 1997) from `scipy.optimize` to perform global optimization. DE is a population-based stochastic optimizer that is robust to local minima and performs well in low-dimensional continuous spaces.

Implementation details. Table 8 summarizes the regression and optimization setup used in the calibration process. This calibration is lightweight and only requires a few warm-up runs on the target hardware to adapt the draft size dynamically for optimal throughput.

G.2 Number of Calibration Samples

In the hyperparameter settings section (Appendix A), we set the default number of calibration samples to 5. Figure 6 analyzes how the size of

Table 8: Regression and optimization setup for throughput calibration.

Component	Configuration
$f_{\text{accept}}(g)$	Polynomial regression using `PolynomialFeatures(degree=3)` with `LinearRegression()`.
$f_{\text{time}}(g)$	B-spline regression using `SplineTransformer(degree=2, n_knots=8)` followed by `LinearRegression()`.
$f_{\text{speed}}(g)$	Defined as the ratio $f_{\text{accept}}(g)/f_{\text{time}}(g)$.
Optimization method	Global optimization using Differential Evolution (DE) (Storn and Price, 1997).
Objective function	Minimize $-f_{\text{speed}}(g)$ to maximize throughput.
Solver configuration	`differential_evolution(lambda x: -g(x[0]), bounds, seed=42, maxiter=1000, atol=1e-6, tol=1e-6)`.
Calibration output	Optimal draft size $g^\star$ yielding maximum estimated throughput.

the calibration set influences both the final optimal draft length and the time required for calibration. The results show that varying the calibration size has little impact on the optimal draft length. We attribute this stability to the average acceptance length and the forward time, which remains nearly unchanged across different calibration sizes. Therefore, instead of using a large calibration set, our method can operate with a relatively small number of samples, reducing calibration time significantly. On an NVIDIA A100 GPU, the calibration process with a small set takes about 800 seconds. Importantly, this calibration is performed only once when deploying on a new hardware environment and can be reused for all subsequent decoding tasks.

H Algorithms

Algorithm 2 Speculative Decoding (Multiple guesses and Greedy Sampling)

Given guess size K, number of guesses G, and target length T.
Given initial prompt sequence $\mathbf{x}$.
while $n < T$ **do**
  Obtain multiple drafts $\tilde{Y} = \{\tilde{y}^{(1)}, \tilde{y}^{(2)}, \ldots, \tilde{y}^{(G)}\}$.
  In parallel, compute $K + 1$ verification tokens y':
  for $i = 1 : K$ **do**
    $y_i'^{(g)} = \arg\max P_M(y_i \mid \tilde{y}_{i-1}^{(g)}, \mathbf{x}), \quad \forall g \in \{1, \ldots, G\}$
  end for
  Identify the sequence $\tilde{y}^{(g^\star)}$ with the highest token matches and the corresponding $y'^{(g)}$.
  for $t = 1 : K$ **do**
    if $y_t'^{(g)} = \tilde{y}_t^{(g^\star)}$ **then**
      Set $y_{n+t} \leftarrow \tilde{y}_t^{(g^\star)}$ and $n \leftarrow n + 1$.
    else
      $y_{n+t} \leftarrow y_t'^{(g)}$ and exit for loop.
    end if
  end for
end while

Algorithm 3 Greedy Verification

Require: sequence $\mathbf{x}$, model P_M, guesses $\mathcal{G} = \{g^i\}$ with $i \in [0, G-1]$
Ensure: o {accepted tokens of length 1 to N}

```
function GreedyVerification(x, P_M, G)
    D ← ∅                                                   ▷ Store the distributions
    V ← G                                                   ▷ Store the current guesses
    for i = 0 to G − 1 do
        D.append(P_M(g'^(i), x_next | g^(i), x))   ▷ Last token of x and g^(i) outputs – total N distributions
    end for
    for i = 1 to N − 1 do
        j ← 1
        is_accept ← 0
        P ← D[1]_i
        while j ≤ size(V) do
            s_j ← V[j]_i
            if s_j = arg max P then   ▷ accepted, update all potential speculations and probabilities
                o.append(s_j)
                is_accept ← 1
                V_new, D_new ← ∅, ∅
                for k = j to size(V) do
                    if s_j = V[k]_i then
                        V_new.append(V[k])
                        D_new.append(D[k])
                    end if
                end for
                V, D ← V_new, D_new
                break
            else                                            ▷ rejected, go to next speculation
                j ← j + 1
            end if
        end while
        if is_accept then
            continue
        else                                                ▷ guarantee one step movement
            o.append(arg max P)
            break
        end if
    end for
    if is_accept then
        o.append(arg max D[1]_N)
    end if
    return o
end function
```

Algorithm 4 Sample Verification

Require: sequence x, model P_M, guesses g^i with $i \in [0, G-1]$
Ensure: o {accepted tokens of length 1 to N}

```
function SAMPLEVERIFICATION(x, P_M, g)
    D ← ∅                                                   ▷ Store the distributions
    V ← 𝒢                                                   ▷ Store the current guesses
    for i = 0 to G − 1 do
        D.append(P_M(g'^(i), x_next | g^(i), x))            ▷ Last token of x and g^(i) outputs – total N distributions
    end for
    for i = 1 to N − 1 do
        j ← 1
        is_accept ← 0
        𝒫_j ← D[j]_i
        while j ≤ size(V) do
            s_j ← V[j]_i
            sample r ~ U(0, 1)
            if r ≤ 𝒫_j(s_j) then        ▷ accepted, update all potential speculations and probabilities
                o.append(s_j)
                is_accept ← 1
                V_new, D_new ← ∅, ∅
                for k = j to size(V) do
                    if s_j = V[k]_i then
                        V_new.append(V[k])
                        D_new.append(D[k])
                    end if
                end for
                V, D ← V_new, D_new
                break
            else                                            ▷ rejected, go to next speculation
                𝒫_j(s_j) ← 0
                𝒫_{j+1} = norm(𝒫_j)
                j ← j + 1
            end if
        end while
        if is_accept then
            continue
        else                                                ▷ guarantee one step movement
            sample x_next ~ 𝒫_j
            o.append(x_next)
            break
        end if
    end for
    if is_accept then
        o.append(sample x_next ~ D[1]_N)
    end if
    return o
end function
```