UNISPEC: Training-Free Speculative Decoding for Robust LLM Acceleration Across Languages and Hardware

Dinh-Truong Do[1*], Nguyen-Khang Le[1*], Nguyen Le Minh[1]

[1]Japan Advanced Institute of Science and Technology

Correspondence: {truongdo, lnkhang, nguyenml}@jaist.ac.jp

Abstract

Speculative decoding accelerates large language model (LLM) inference through a draft-and-verify paradigm, yet existing methods face three key limitations: reliance on fixed draft templates that ignore device-specific verification costs, lack of mechanisms to assess draft token quality, and suboptimal tree expansion strategies. We introduce UNISPEC, a training-free, lossless speculative decoding framework that enables robust, plug-and-play LLM acceleration across diverse hardware configurations and languages. UNISPEC incorporates three novel components: (1) a device-aware calibration mechanism that determines the optimal draft size by measuring the acceptance-time trade-off on each target device; (2) a confidence score estimation module that assigns quality scores to n-grams based on the verifier's token probabilities, enabling selective retention of high-quality draft candidates; and (3) an improved tree expansion strategy that broadens first-level exploration and applies threshold-based filtering to prune low-confidence nodes. To comprehensively evaluate multilingual performance, we create a comprehensive benchmark, covering seven languages across seven generation tasks. Experiments with various LLM architectures, hardware environments, and languages demonstrate that UNISPEC consistently outperforms existing training-free methods, achieving speedups of up to 2.6× while maintaining output quality identical to standard autoregressive decoding. Our code and benchmark are publicly available.

1 Introduction

Generating long sequences with low latency using Large Language Models (LLMs) is a critical requirement, especially given recent trends in 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 that scales 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),

[*]These authors contributed equally to this work

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 6288–6310
July 2-7, 2026

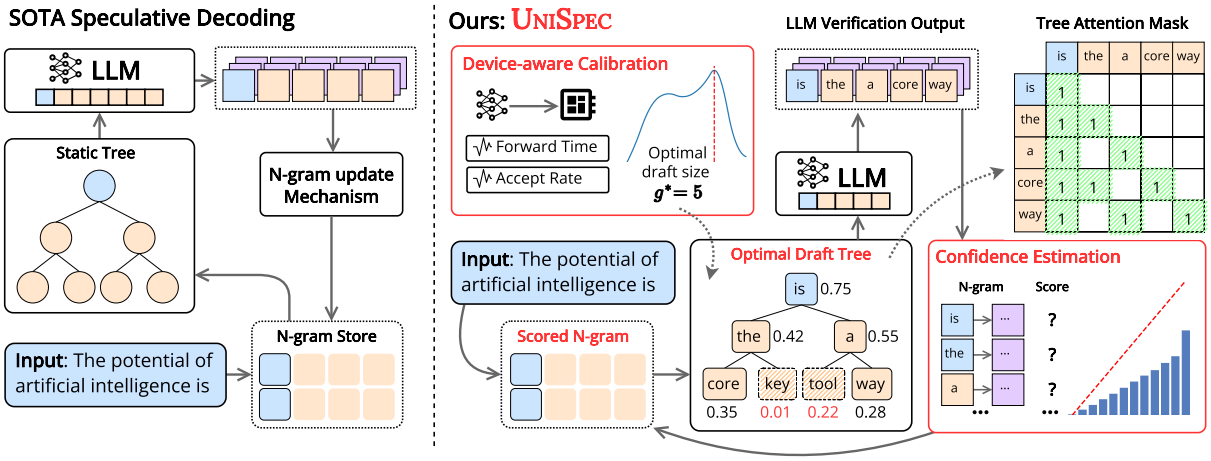

Figure 1: Overview of UniSpec compared with other SOTA approaches. Newly introduced components are highlighted in red. blue boxes indicate the input tokens, orange boxes indicate the draft tokens, purple boxes indicate top next-token predictions of the LLM.

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 hardware capacity, leading to inconsistent speedups across different hardware configurations. 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 UniSpec (Figure 1, Right), a training-free speculative decoding framework for robust, plug-and-play LLM acceleration across hardware and languages. Upon deployment, UniSpec 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, UniSpec 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 — Llama-3-8B-Instruct (Dubey et al., 2024) and Qwen-3-8B/14B (Yang et al., 2025) — show that UniSpec 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 UniSpec, a *training-free* speculative decoding framework that enables robust, plug-and-play acceleration of large language models across hardware and languages.
- UniSpec introduces **(1)** a *hardware calibration mechanism* to derive the optimal draft size for each device, ensuring consistent speedup; **(2)** a *confidence-guided tree construction strategy* that scores and filters n-grams based on model probabilities; and **(3)** *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

Autoregressive Decoding. Given an input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of length n, 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 n-th token, conditioned on all preceding tokens, is given by $P_M(x_n \mid \mathbf{x}_{1:n-1})$. The next token x_n 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 $x_n = \text{argmax}\, P_M(x_n \mid \mathbf{x}_{1:n-1})$. 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 $y_i = \text{argmax}\, P_M(y_i \mid y_{1:i-1}, \mathbf{x})$.

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.

Training-free N-gram Strategies. Several prior works employ trained drafter models (Chen et al., 2023; Leviathan et al., 2023; Cai et al., 2024) or modify the original LLM via self-speculative decoding (Elhoushi et al., 2024; Li et al., 2024b). While effective, these approaches require costly training and may alter the model's capabilities. Training-free n-gram strategies instead generate draft tokens directly from stored sequences without additional training. Common retrieval strategies include 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 combinations thereof (Le et al., 2025). See Appendix B for details.

3 UniSpec Decoding

UniSpec 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, UniSpec 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.

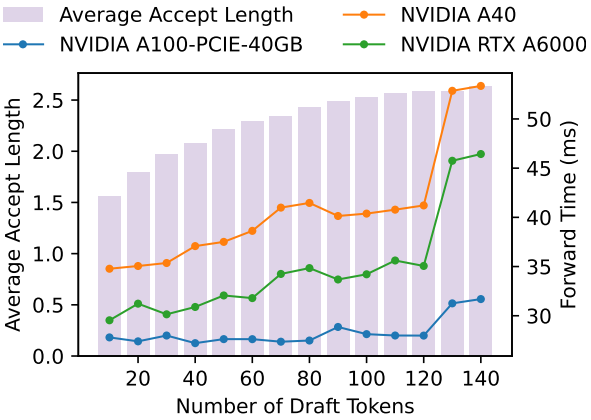

Figure 2: Average acceptance length and forward time by the number of draft tokens (Llama-3-8B-Instruct).

Device-aware Calibration. 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. Let g denote the number of drafted tokens (draft size), and let $s(g)$ and $\tau(g)$ denote the verification latency and average acceptance length, respectively, as functions of g. Following standard practice in speculative decoding, the draft size g is determined before generation and remains fixed throughout the inference process. The two quantities $s(g)$ and $\tau(g)$ exhibit opposing trends with respect to draft size: increasing g provides more candidate tokens for verification, raising the probability of accepting longer sequences and thus increasing $\tau(g)$; however, larger g also lengthens the input sequence for verification, increasing the computational cost $s(g)$. This interplay creates a nontrivial optimization landscape where performance depends on balancing these opposing effects.

Empirically, $\tau(g)$ remains relatively stable across hardware, while $s(g)$ varies substantially with GPU architecture (Figure 2). Consequently,

the optimal draft size is device-dependent. The throughput can be expressed as:

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

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

where N_{step} is the number of forward steps. This formulation is valid under two standard assumptions in speculative decoding: **(i)** verification latency dominates the total step time, as drafting with lightweight models or cached representations incurs negligible overhead compared to target model inference; and **(ii)** system overheads independent of the draft size g (e.g., memory allocation, scheduling) remain approximately constant across configurations. Under these assumptions, optimizing throughput reduces to optimizing the ratio $\tau(g)/s(g)$.

Prior work (Kwon et al., 2023; Zheng et al., 2024) has shown that GPU scheduling, KV-cache bandwidth, and parallelism are highly hardware-dependent and thus difficult to capture with analytic performance models. As a result, empirical performance modeling is commonly adopted in practice. Following this paradigm, we directly measure end-to-end latency under different configurations, fit practical regression models, and optimize based on the observed behavior. Specifically, we perform m calibration runs with draft sizes $g_1, \ldots, g_m \in [g_{\min}, g_{\max}]$, measuring s_i and τ_i for each, yielding datasets $S = \{(g_i, s_i)\}$ and $T = \{(g_i, \tau_i)\}$. We fit regression models:

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

and solve for the optimal draft size:

$$g^\star = \text{argmax}_{g \in [g_{\min}, g_{\max}]} \frac{f_{\text{accept}}(g)}{f_{\text{time}}(g)} \quad (4)$$

using Differential Evolution (Storn and Price, 1997). By measuring on the target hardware, our calibration captures device-specific constraints and parallelization characteristics, ensuring $g^\star$ reflects true deployment performance.

The calibration procedure is shown in Algorithm 1 (lines 3–11). Details are in Appendix F; empirical validation of Equation 2 is in Appendix G.1; comparison with simple calibration, such as grid-search, is detailed in Appendix G.2.

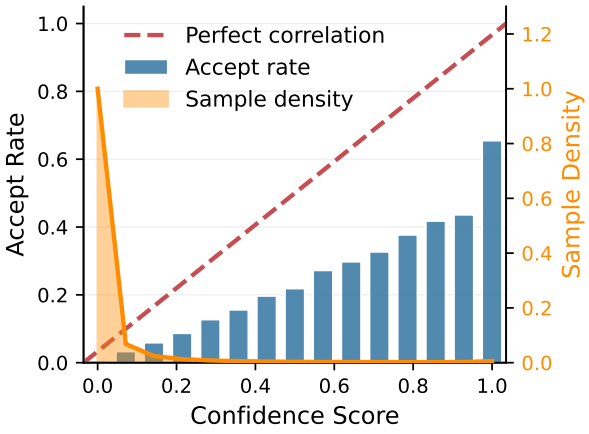

Figure 3: Acceptance rate at different confidence scores, measured with Llama-3-8B-Instruct. Sample density indicates the probability that a node with a specific confidence score appears in the draft tree.

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.

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

Algorithm 1 UNISPEC 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 have 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: $\mathbf{x} \leftarrow \mathbf{x} \oplus \text{hits}$
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 $\mathbf{y} = \mathbf{x}_{n+1:n+m}$

expansion, improving efficiency (Alg. 1, line 28).

Although EAGLE-2/3 (Li et al., 2025) also prunes the draft tree based on confidence scores, the underlying approach is fundamentally different. EAGLE-2/3 is training-based and relies on a learned drafter model to generate candidate tokens and to provide confidence estimates for pruning. In contrast, UNISPEC is entirely training-free and generates candidates from an N-gram store, which initially offers no intrinsic confidence signal. This gap is bridged by the confidence score estimation module of UNISPEC, enabling effective tree expansion and pruning.

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, retrieval-augmented generation, and code generation, each with 80 samples per subtask. In addition, we include the HumanEval benchmark (Chen et al., 2021), which focuses on code generation; we use a subset of 80 entries to match the size of other subtasks.

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 UNISPEC and baseline methods across different models and hardware on Spec-Bench tasks. Across all settings, UNISPEC consistently achieves the strongest speedups, substantially outperforming prior training-free methods. Notably, our method maintains robust speedups even on resource-constrained GPUs 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 across heterogeneous hardware. In the per-task breakdown, UNISPEC ranks first across all evaluated tasks.

We also report the average acceptance length (τ). Note that τ alone does not determine speedup, since larger draft sizes raise τ but slow each forward pass. Nonetheless, our method consistently achieves higher τ than baselines due to our dynamic speculation tree with confidence-based filtering and enhanced first-level expansion.

Multilingual robustness. Figure 4 presents the average performance of Multi-SpecBench across all languages on four hardware configurations (detailed results in Appendix D). UNISPEC achieves the best cross-language averages across all testbeds, consistently outperforming all baselines. Notably, on lower-resource GPUs (RTX A6000 and RTX 3090), the performance gap between UNISPEC and baselines widens—previous methods are primarily tuned for high-end GPUs, whereas our approach adapts effectively to different hardware. Results on Qwen models further confirm consistent gains across architectures.

5.2 Analysis

Ablation Study. We analyze the contribution of each UNISPEC component using LLaMA-3-8B-Instruct on an RTX 3090 with Spec-Bench (Table 2). Removing *device-aware calibration* (using fixed draft length) substantially reduces speedup, showing that fixed configurations fail to generalize across hardware. Removing *score-based tree-node estimation* further degrades performance, indicating that confidence-weighted branching yields stronger draft candidates. Disabling the *tree expansion mechanism* reduces both speedup and acceptance length, confirming the value of enhanced first-level exploration and threshold-based filtering. Despite these degradations, all variants remain comparable to existing baselines, highlighting the complementary strengths of each component.

Comparison to training-based methods. We compare UNISPEC with training-based methods EAGLE-2 (Li et al., 2024b) and EAGLE-3 (Li et al., 2025), which train auxiliary drafters using the target model's hidden states. As shown in Table 3, UNISPEC achieves competitive performance on English and significantly outperforms EAGLE methods on non-English languages, where the trained drafter struggles with cross-lingual generalization. This demonstrates the strength of our training-free framework, which achieves robust multilingual efficiency without retraining. See Appendix E.4 for more analysis.

Optimal values of draft tokens. Figure 5 shows the device-specific optima $g^\star$ selected by our calibration, confirming that optimal draft length is *not universal* but strongly tied to verification cost. On memory-constrained consumer GPUs, smaller draft sizes maximize τ/s by reducing verification overhead. In contrast, data-center GPUs sustain higher parallelism with longer inputs, enabling larger optimal draft sizes. This variation across environments underscores the necessity of device-aware calibration: acceptance gains from larger drafts saturate quickly, and without tuning, extra draft tokens can inflate latency enough to reduce overall speed.

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.
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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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 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).

Other Analysis. Detailed results for all languages appear in Appendix D. Additional evaluations under different sampling temperatures are in Appendix E.1, multi-GPU performance in Appendix E.2, calibration size/time analysis in Appendix F.2, analysis on different batch sizes in Appendix E.3, lossless property in Appendix E.5, long-context evaluation in Appendix E.6, sensitivity to the confidence threshold in Appendix E.7, and sensitivity to draft-size deviation in Appendix E.8.

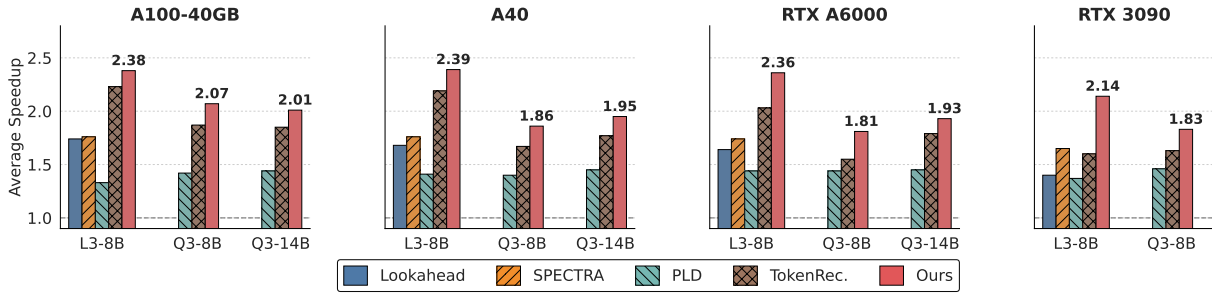

Figure 4: Average speedup across languages of Multi-SpecBench for each testbed (S1–S4).

Method	MT Bench Spd.	MT Bench τ	Code Spd.	Code τ
SPECTRA	1.25	1.96	1.32	1.98
TokenRec.	1.51	2.57	1.47	2.68
UNISPEC (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 UNISPEC on RTX 3090 (Llama-3-8B-Instruct, Spec-Bench) for multi-turn conversation and code generation tasks.

Method	EN Spd.	EN τ	JA Spd.	JA τ	DE Spd.	DE τ
EAGLE-2	1.86	3.86	0.99	2.05	1.09	2.27
EAGLE-3	**2.38**	**5.10**	0.71	1.44	1.10	2.41
UNISPEC (ours)	1.97	2.99	**2.72**	**4.23**	**2.41**	**3.68**

Table 3: Comparison with training-based methods on Multi-SpecBench for English (EN), Japanese (JA), and German (DE) using LLaMA-3.1-8B-Instruct.

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.

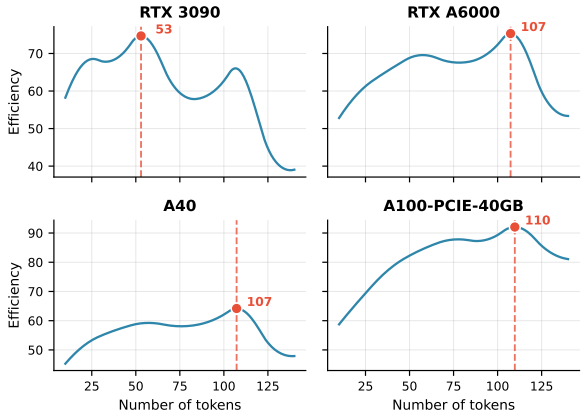

Figure 5: Calibrated optimal draft size $g^\star$ by environment settings (S1–S4).

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 UNISPEC, 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, UNISPEC 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 UNISPEC provides practical, 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., Chinese, 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 UNISPEC 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.

Acknowledgements

This work was partly supported by Japan Science and Technology Agency (JST) as part of Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE), Grant Number JPMJAP25B2 and JST CREST, Japan, Grant Number JPMJCR2554. This work was also partly conducted in collaboration with Ricoh Company, LTd. The authors sincerely appreciate their insightful discussions and continuous support.

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*.

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.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pages 611–626, New York, NY, USA. Association for Computing Machinery. Event-place: Koblenz, Germany.

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.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

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.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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).

UniSpec 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 $\{0.01, 0.025, \mathbf{0.05}, 0.1, 0.2\}$*. 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 F.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

*Bold values indicate the best result on the calibration data

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 generate 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).

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).

D Detailed Multilingual Benchmark Results

Table 5 reports the complete results for speedup and acceptance length across all tasks in Spec-Bench, as well as per-language performance for the multilingual Multi-SpecBench introduced in Section 5.1. Across all languages and tasks, UNISPEC 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 UNISPEC to diverse tasks, languages, and hardware configurations.

E Additional Experiments and Analysis

E.1 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-Instruct using Spec-Bench, as shown in Table 6. We observe slight performance degradation at higher sampling temperatures, likely due to lower acceptance rates during the sampling-based verification phase, consistent with prior findings (Luo et al., 2025; Le et al., 2025). Despite this, the speedup ratio of UNISPEC remains consistently around 2.0×, demonstrating sustained acceleration and confirming the robustness of our approach across different decoding strategies.

E.2 Multi-GPU / Distributed Setting

A critical consideration for practical deployment is how UNISPEC 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 UNISPEC 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, UNISPEC still consistently outperforms baseline training-free methods under distributed inference, demonstrating that its core acceleration benefits remain effective even when scaling beyond a single GPU.

E.3 Batch Size Analysis

Large Language Model (LLM) inference is typically memory-bound during the decoding phase, particularly when the batch size is small (Miao et al., 2024; Fu et al., 2024). In this regime, the GPU compute units are often underutilized while waiting for data to be fetched from High Bandwidth Memory (HBM). Speculative decoding exploits these idle compute cycles to verify multiple draft tokens in parallel without significantly increasing wall-clock latency. However, in real-world serving scenarios where request batching is employed to maximize throughput, the hardware utilization dynamics shift. As batch size increases, the computational intensity of the workload rises, gradually shifting the system from a memory-bound to a compute-bound regime. Under such conditions, the benefits of speculative decoding may diminish due to increased contention for GPU compute resources. In this section, we examine the robustness of UNISPEC under varying batch sizes to assess its scalability in high-throughput serving environments.

We evaluated UNISPEC on an NVIDIA A100 (40GB) using Llama-3-8B-Instruct with FP16 precision, varying the batch size from 1 to 5. As illus-

Model	Method	DE Spd.	DE τ	ES Spd.	ES τ	FR Spd.	FR τ	JA Spd.	JA τ	VI Spd.	VI τ	ZH Spd.	ZH τ	AVG 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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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
	UNISPEC (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**).

trated in Table 8, UNISPEC maintains a significant speedup of $2.10\times$ at batch size 1. As the batch size increases, the speedup ratio exhibits a gradual decline, dropping to $1.73\times$ at a batch size of 5. This trend is consistent with prior findings in speculative decoding literature (Li et al., 2024c;

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	Autoregressive	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	Autoregressive	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**).

Batch size	**1**	**2**	**3**	**4**	**5**
Speedup	2.10×	2.07×	2.03×	1.90×	1.73×

Table 8: Speedup ratios of UNISPEC across increasing batch sizes on an NVIDIA A100 (40GB) using Llama-3-8B-Instruct (FP16). The evaluation was conducted on the MT-bench dataset with temperature set to 0.

Cai et al., 2024). The reduction in speedup occurs because the verification of draft trees for multiple concurrent sequences saturates the GPU's streaming multiprocessors (SMs), increasing the latency of the forward pass. Despite this decline, UNISPEC continues to provide substantial acceleration over autoregressive decoding even at higher batch sizes. This suggests that our device-aware calibration, while primarily optimized for latency in single-stream scenarios, effectively identifies a draft size that remains beneficial even as compute contention increases.

E.4 More comparison with EAGLE family

Table 9 shows the full comparison with the EAGLE family of training-based speculative decoding methods using Llama3-8B-Instruct and Llama3.1-8B-Instruct.

E.5 UNISPEC Being Lossless

Theoretical basis. The draft-and-verify paradigm used in UNISPEC has been rigorously proven to reproduce exactly the same outputs as standard autoregressive decoding in prior work (Miao et al., 2024; Fu et al., 2024). This ensures that the generation trajectory is identical under any draft size or verification order.

Empirical verification. Across all of our experiments, we explicitly compared the outputs generated by UNISPEC with those from standard autoregressive decoding. Prior work (Fu et al., 2024; Le et al., 2025) has shown that under float32 precision, speculative decoding produces outputs that are identical to those from autoregressive decoding, and under float16, differences are limited to only

Method	LLaMA-3-8B-Instruct						LLaMA-3.1-8B-Instruct					
	EN		JA		DE		EN		JA		DE	
	Spd.	τ	Spd.	τ	Spd.	τ	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	1.86	3.86	0.99	2.05	1.09	2.27
EAGLE-3	–	–	–	–	–	–	**2.38**	**5.10**	0.71	1.44	1.10	2.41
UNISPEC (ours)	2.13	2.94	**2.45**	**3.40**	**2.21**	**3.08**	1.97	2.99	**2.72**	**4.23**	**2.41**	**3.68**

Table 9: Full comparison with training-based speculative decoding methods on Multi-SpecBench for English (EN), Japanese (JA), and German (DE). Results for EAGLE-1, EAGLE-2, and EAGLE-3 are reproduced using the official checkpoints released in their original repositories. The symbol '–' indicates that the EAGLE checkpoint for the corresponding LLM is not publicly available.

a few tokens due to numerical precision effects. To further verify fidelity, we assess the generation quality of UNISPEC and autoregressive outputs on the CNN and GSM8K datasets, measured in ROUGE scores. We present the results in Table 10.

E.6 Long-Context Evaluation

A natural concern for retrieval-based speculative decoding is whether the benefits persist under long-context generation, where KV-cache pressure and draft-tree bookkeeping may erode the observed speedups. To evaluate this, we run UNISPEC with Qwen-3-8B on an RTX A6000 across all Spec-Bench tasks, sweeping the context window from 1K up to 32K tokens.

Table 11 shows that both the average speedup and the average acceptance length τ remain essentially flat as the context grows by more than an order of magnitude. This stability is expected: UNISPEC's draft-tree construction operates over the n-gram store and is independent of the KV-cache length, while the verification cost grows sublinearly with context size. We therefore conclude that the method retains its acceleration benefits under long-context memory pressure.

E.7 Sensitivity to the Confidence Threshold r.

The confidence threshold r governs how aggressively the tree expansion module prunes low-confidence nodes (Alg. 1, line 28). We examine its effect by sweeping $r \in \{0.0, 0.01, 0.025, 0.05, 0.1, 0.2\}$ on Llama-3-8B / RTX 3090, where r=0.0 corresponds to no pruning. Table 12 reports the results.

Applying a modest threshold consistently improves the average speedup over the no-pruning baseline, peaking at r=0.05 (1.99× average speedup vs. 1.90× at r=0.0). Performance degrades only mildly as r is increased further, and never falls below the r=0.0 baseline within the swept range. Together with Figure 3, which shows that low-confidence tokens are densely distributed yet exhibit consistently low acceptance rates, this confirms that threshold pruning primarily removes low-recall, low-utility candidates rather than acceptable tokens, and therefore does not cap the achievable speedup in practice.

E.8 Sensitivity to Draft-Size Deviation

A practical concern is how much the selected draft size $g^\star$ may deviate from its optimum before the speedup benefits are lost. We investigate this on NVIDIA A100 with Llama-3-8B, for which the calibrated optimum is $g^\star$=110. We evaluate UNISPEC with fixed draft sizes in the neighborhood $g \in \{95, 100, 105, 110, 115, 120, 125\}$. Table 13 reports the results.

Performance is remarkably insensitive to small deviations from $g^\star$. Within a ± 15 token window, the average speedup drops only from 2.13× at the optimum to 1.95× at the edges, which still substantially exceeds autoregressive decoding. Combined with the online recalibration strategy described in §3, this flat neighborhood explains why UNISPEC remains robust even when the effective hardware performance drifts over time.

F Calibration Details

F.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.

Precision	Dataset	Method	ROUGE-1	ROUGE-2	ROUGE-L	Speedup	τ
FLOAT32	CNN	Autoregressive	9.90	4.14	6.95	1.00	1.00
		UNISPEC	9.90	4.14	6.95	2.09	2.89
	GSM8K	Autoregressive	1.82	0.41	1.67	1.00	1.00
		UNISPEC	1.82	0.41	1.67	2.24	3.06
FLOAT16	CNN	Autoregressive	9.98	4.36	7.06	1.00	1.00
		UNISPEC	10.05	4.37	7.08	2.12	2.93
	GSM8K	Autoregressive	1.81	0.41	1.65	1.00	1.00
		UNISPEC	1.65	0.40	1.58	2.26	3.10

Table 10: Evaluation of UNISPEC decoding under float32 and float16 precision on CNN/DailyMail and GSM8K. ROUGE scores, speedups over autoregressive decoding, and average acceptance length (τ) are reported for LLaMA-3-8B-Instruct.

Context	MT		Trans		Sum		QA		Math		RAG		Code		Avg.
	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.
1024	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
2048	1.83	2.65	2.03	2.78	1.86	2.66	1.96	2.73	2.09	2.88	1.90	2.75	1.88	2.65	1.93
4096	1.81	2.69	2.04	2.79	1.86	2.65	1.96	2.82	2.08	2.88	1.87	2.70	1.88	2.71	1.93
8192	1.81	2.68	2.03	2.79	1.87	2.67	1.94	2.85	2.09	2.91	1.87	2.69	1.88	2.78	1.93
16384	1.81	2.74	2.04	2.79	1.85	2.65	1.93	3.18	2.08	2.90	1.87	2.70	1.86	2.90	1.92
32768	1.82	2.68	2.04	2.78	1.87	2.67	1.92	3.33	2.06	2.97	1.88	2.71	1.88	3.65	1.93

Table 11: UNISPEC performance under increasing context length (Qwen-3-8B, RTX A6000). Both speedup and acceptance length τ remain stable from 1K up to 32K tokens.

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 14 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.

F.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 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

r	MT		Trans		Sum		QA		Math		RAG		Code		Avg.
	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.
0.0	1.87	2.40	1.82	2.31	1.82	2.38	1.79	2.27	2.06	2.59	2.17	2.76	1.77	2.39	1.90
0.01	1.89	2.55	1.85	2.43	1.86	2.56	1.85	2.43	2.10	2.74	2.15	2.89	1.80	2.55	1.93
0.025	1.92	2.61	1.87	2.48	1.88	2.61	1.87	2.46	2.11	2.76	2.16	2.97	1.83	2.62	1.95
0.05	**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**
0.1	1.91	2.57	1.88	2.46	1.88	2.59	1.88	2.46	2.13	2.77	2.18	2.95	1.81	2.59	1.95
0.2	1.87	2.46	1.84	2.37	1.84	2.46	1.83	2.34	2.08	2.66	2.16	2.88	1.77	2.47	1.91

Table 12: Sensitivity of UNISPEC to the confidence threshold r (Llama-3-8B, RTX 3090). r=0.0 disables pruning. A mild threshold consistently improves speedup, peaking at r=0.05.

Draft size g	MT		Trans		Sum		QA		Math		RAG		Code		Avg.
	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.	τ	Spd.
95	1.90	2.93	1.88	2.82	1.89	2.98	1.85	2.74	2.13	3.10	2.11	3.31	1.93	2.98	1.95
100	1.99	2.80	1.92	2.71	2.05	2.85	1.92	2.67	2.14	2.98	2.32	3.22	1.97	2.84	2.04
105	2.04	2.76	1.92	2.64	2.05	2.78	1.96	2.65	2.21	2.97	2.38	3.21	2.00	2.79	2.08
110 ($g^\star$)	**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**
115	2.06	2.83	1.99	2.72	2.09	2.85	1.96	2.68	2.23	3.02	2.35	3.19	2.06	2.86	2.11
120	2.00	2.80	1.90	2.68	2.03	2.82	1.91	2.66	2.16	2.98	2.28	3.15	1.95	2.81	2.03
125	1.90	2.95	1.87	2.79	1.87	2.95	1.86	2.77	2.11	3.07	2.12	3.31	1.93	2.98	1.95

Table 13: Sensitivity of UNISPEC to deviations from the calibrated optimal draft size $g^\star$=110 (Llama-3-8B, NVIDIA A100). Speedup degrades only mildly within a $\pm$15 token window and remains well above the autoregressive.

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.

Table 14: Regression and optimization setup for throughput calibration.

length and the forward time, which remain 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 (approximately 13 minutes). Importantly, this calibration needs to be performed only once when deploying on a new hardware environment and can be reused for all subsequent decoding tasks.

G Detailed Analysis on Device-calibration

G.1 Empirical Validation of Efficiency Formula

To empirically validate the modeling in Equation 2, Table 15 shows the measured $\tau(g)$ and $s(g)$, along with the expected and real-world throughput for different draft sizes. The results show that the expected throughput modeled by our formula (Equation 2) reflects real-world throughput, with only minor deviations due to hardware-dependent effects (GPU scheduling, KV-cache memory bandwidth, and parallelism behavior, etc.). This supports the validity of the modeling in Equation 2.

G.2 Device-calibration versus grid-search

Comparing the device calibration module against a common search strategy, such as grid search, can further validate the effectiveness of our device-aware calibration module for determining the optimal draft size. To this end, we performed a grid search with a step size of 25 and compared its selected draft sizes against the estimate produced by our method. As shown in Table 16, the grid search identifies local optima at 100 and 125 tokens. However, under the same number of calibration queries, our regression-based method estimates an optimal

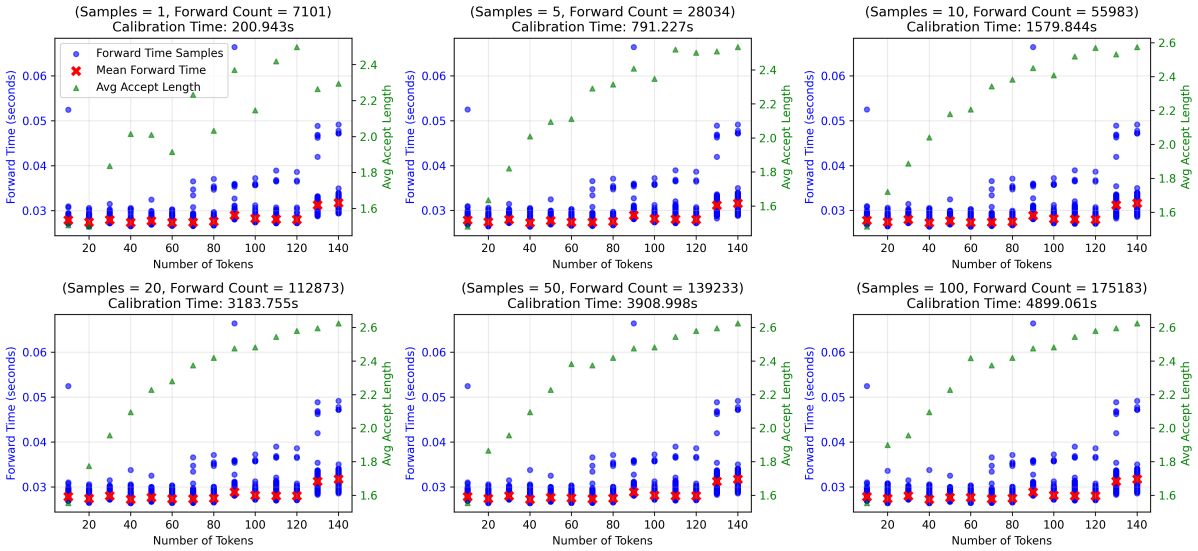

Figure 6: Forward time, average accept length, and calibration time when calibrating with different numbers of samples.

Draft size	Average $\tau(g)$ (tokens)	Average $s(g)$ (seconds)	Expected TP (tok/s)	Real TP (tok/s)
$g = 75$	2.70	0.0275	99.66	87.68
$g = 111$ (optimal)	2.94	0.0281	104.65	90.53

Table 15: Measured $\tau(g)$ and $s(g)$, and expected vs. real-world throughput.

draft size of $g = 111$, which yields higher throughput. These results highlight the benefit of our continuous optimization formulation, which can better capture the true performance optimum than discrete grid exploration.

The key difference is that grid search optimizes the draft size only over a fixed discrete set (e.g., steps of 25), so the best result is limited by that granularity. Reducing the step size improves precision but dramatically increases the number of evaluations, making grid search costly on large models or slower devices. In contrast, our regression-guided optimization (via differential evolution) efficiently estimates the peak throughput in a continuous manner, achieving higher-resolution draft size selection.

G.3 Generalization of device-calibration module

UNISPEC's Device-Aware Calibration module is model-agnostic and strategy-agnostic, and can be applied to any existing speculative decoding method to identify the draft size that is optimal for the corresponding drafting strategy. To demonstrate this, Table 17 reports the speedup achieved using both the default draft size and the optimal draft size selected by our device-aware calibration across multiple drafting strategies. The results show that the calibration module of UNISPEC generalizes well across diverse speculative decoding methods following the *draft-and-verify* paradigm, yielding substantial improvements in decoding speed.

H Algorithms

Draft size	Throughput (tok/s)	Average Accept Length τ
$g = 25$	72.17	2.24
$g = 50$	81.93	2.56
$g = 75$	86.68	2.74
$g = 100$	89.49	2.83
$g = 125$	89.49	**2.94**
$g = 111$ (optimal)	**90.53**	2.90

Table 16: Comparison between grid search and the device-calibration module for finding the optimal draft size.

		MT Bench		**Code**	
Method	**Draft size**	Spd.	τ	Spd.	τ
Lookahead	120	1.15	2.00	1.11	2.01
Lookahead + Device-Aware	60 (optimal)	**1.40**	1.85	**1.44**	1.98
SPECTRA	60	1.25	1.96	1.32	1.98
SPECTRA + Device-Aware	30 (optimal)	**1.47**	1.77	**1.66**	1.76
Token Recycling	80	1.51	2.57	1.47	2.68
Token Recycling + Device-Aware	53 (optimal)	**1.78**	2.33	**1.75**	2.41

Table 17: Comparison of Lookahead, SPECTRA, and Token Recycling under standard vs. device-aware configurations on RTX 3090 (Llama-3-8B-Instruct, Spec-Bench) for MT Bench and code generation. Results are reported as speedup relative to autoregressive decoding (Spd.) and average acceptance length (τ).

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($\mathbf{x}, P_M, \mathcal{G}$)
  $D \leftarrow \emptyset$ ▷ Store the distributions
  $V \leftarrow \mathcal{G}$ ▷ Store the current guesses
  for $i = 0$ to $G-1$ **do**
    D.append($P_M(g'^{(i)}, x_{\text{next}} | g^{(i)}, \mathbf{x})$) ▷ Last token of $\mathbf{x}$ and $g^{(i)}$ outputs – total N distributions
  end for
  for $i = 1$ to $N-1$ **do**
    $j \leftarrow 1$
    is_accept $\leftarrow 0$
    $\mathcal{P} \leftarrow D[1]_i$
    while $j \leq$ size(V) **do**
      $s_j \leftarrow V[j]_i$
      if $s_j = \arg\max \mathcal{P}$ **then** ▷ accepted, update all potential speculations and probabilities
        o.append(s_j)
        is_accept $\leftarrow 1$
        $V_{\text{new}}, D_{\text{new}} \leftarrow \emptyset, \emptyset$
        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
        $V, D \leftarrow V_{\text{new}}, D_{\text{new}}$
        break
      else ▷ rejected, go to next speculation
        $j \leftarrow j + 1$
      end if
    end while
    if is_accept **then**
      continue
    else ▷ guarantee one step movement
      o.append$(\arg\max \mathcal{P})$
      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)
  $D \leftarrow \emptyset$ ▷ Store the distributions
  $V \leftarrow \mathcal{G}$ ▷ Store the current guesses
  for $i = 0$ to $G-1$ **do**
    D.append($P_M(g'^{(i)}, x_{\text{next}}|g^{(i)}, \mathbf{x})$) ▷ Last token of **x** and $g^{(i)}$ outputs – total N distributions
  end for
  for $i = 1$ to $N-1$ **do**
    $j \leftarrow 1$
    is_accept $\leftarrow 0$
    $\mathcal{P}_j \leftarrow D[j]_i$
    while $j \leq$ size(V) **do**
      $s_j \leftarrow V[j]_i$
      sample $r \sim U(0,1)$
      if $r \leq \mathcal{P}_j(s_j)$ **then** ▷ accepted, update all potential speculations and probabilities
        o.append(s_j)
        is_accept $\leftarrow 1$
        $V_{\text{new}}, D_{\text{new}} \leftarrow \emptyset, \emptyset$
        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
        $V, D \leftarrow V_{\text{new}}, D_{\text{new}}$
        break
      else ▷ rejected, go to next speculation
        $\mathcal{P}_j(s_j) \leftarrow 0$
        $\mathcal{P}_{j+1} = \text{norm}(\mathcal{P}_j)$
        $j \leftarrow j+1$
      end if
    end while
    if is_accept **then**
      continue
    else ▷ guarantee one step movement
      sample $x_{\text{next}} \sim \mathcal{P}_j$
      o.append(x_{next})
      break
    end if
  end for
  if is_accept **then**
    o.append(sample $x_{\text{next}} \sim D[1]_N$)
  end if
  return o
end function