An Empirical Study of Speculative Decoding for Small Language Models

Luca Mainardi
Eindhoven University of Technology
l.mainardi@student.tue.nl

Selcuk Sandikci
NXP Semiconductors
selcuk.sandikci@nxp.com

Joaquin Vanschoren
Eindhoven University of Technology
j.vanschoren@tue.nl

Abstract

Speculative decoding has emerged as a promising approach to accelerate Large Language Model inference. However, existing research has predominantly focused on 7B-70B parameters models, leaving a critical knowledge gap for small language models (1-2B parameters) that are increasingly important for edge computing and agentic AI systems. This paper presents the first comprehensive empirical study of speculative decoding techniques for small language models. We evaluate five distinct method categories across three representative model families and reveal that drafting overhead, rather than draft quality, becomes the primary bottleneck fundamentally limiting acceleration of small models. We demonstrate that traditional independent drafting fails completely due to the suboptimal architecture of available drafters, while self-drafting methods achieve meaningful acceleration only when employing sufficiently efficient draft modules. In contrast, retrieval-based methods with negligible computational overhead yield consistent gains. Based on these insights, we establish practical guidelines for effective small model acceleration.

1 Introduction

Large Language Models (LLMs) generate text autoregressively. Each step requires a complete forward pass through the model and the transfer of all model parameters from the High-Bandwidth Memory to the on-chip cache of modern accelerators like GPUs, creating a memory bandwidth bottleneck (Shazeer, 2019) where inference latency is dominated by memory operations rather than computation.

Speculative decoding addresses this limitation through a draft-then-verify paradigm (Stern et al., 2018; Leviathan et al., 2022; Chen et al., 2023): an efficient draft model generates candidate tokens, which the target model (LLM to be accelerated) verifies in parallel via a single forward pass. When the target model's predictions align with the drafted tokens, multiple tokens can be accepted simultaneously and added to the final generation; otherwise, verification stops at the first mismatch, thus guaranteeing equivalent generation quality.

The speedup of speculative decoding can be approximated as:

$$\text{Speedup} = \frac{1 + \mathbb{E}[\#\text{ accepted tokens}]}{1 + C_d/C_v} \tag{1}$$

where C_d is the drafting cost, C_v is the verification cost (approximately one forward pass of the target model), and accepted tokens refer to draft tokens that pass verification per speculative decoding iteration (one complete draft-then-verify cycle). This formula reveals why speculative decoding has been highly effective for large models (7B-70B parameters): the expensive memory transfers required for large model inference create high C_v, making the C_d/C_v ratio small and thus easily amortized by accepting even a few draft tokens.

However, this success does not directly transfer to small models (1-2B parameters). Despite their increasing importance for edge computing and resource-constrained environments—where on-device inference provides energy efficiency, preserves privacy, and operates without internet connectivity (Zhang et al., 2024)—and their particular promise for agentic AI systems (Belcak et al., 2025), small language models have received little attention in speculative decoding research. This creates a critical knowledge gap: existing techniques, developed and optimized exclusively for large models, may be fundamentally unsuitable for small model acceleration. The fundamental challenge is that for small models, memory transfers are already significantly faster and cheaper — C_v is small. Consequently, the C_d/C_v ratio in Equation 1 becomes dominant rather than negligible. Even modest drafting overhead can eliminate potential speedups, shifting the optimization focus from maximizing draft quality (the primary concern for large models) to minimizing the drafting overhead (C_d/C_v ratio) while maintaining sufficient number of accepted tokens.

To address this gap, we conduct a systematic empirical study of speculative decoding techniques for small language models (1-2B parameters), examining five distinct method categories across three representative model families. Our work makes the following contributions:

Failure of Independent Drafting : We are the first to empirically demonstrate that independent drafting with family-matched models slows down small target model inference due to the lack of sufficiently efficient draft models in the current open-source ecosystem.

Framework for Small Language Model (SLM) Acceleration : We move beyond the conventional "draft quality" focus relevant for large models and establish a new framework for SLMs centered on the trade-off between drafting overhead and draft quality. Through extensive experiments, we reveal that effective small model acceleration requires either negligible drafting costs or sufficiently high drafting quality to justify overhead, achievable only through model-specific training.

Practical Design Principles : We synthesize our findings into practical design principles and a decision framework for engineers and researchers deploying SLMs: (1) Avoid external drafters unless they employ wide-shallow architectures designed for low latency; (2) If training budget is available, leverage model-specific training to achieve draft quality sufficient to overcome computational overhead; (3) Prioritize retrieval-based methods with negligible computational cost when training is not an option; (4) Integrate tree verification mechanisms, when applicable, to enhance performance.

2 Related Work

The formal framework of speculative decoding was established by two seminal works: Speculative Sampling (Leviathan et al., 2022) and its concurrent development (Chen et al., 2023), which demonstrated that smaller draft models can generate candidate tokens verified in parallel by the target model while maintaining equivalence to standard autoregressive generation across diverse sampling strategies. From these foundational works, various methods and variants have been explored, creating a rich landscape of techniques.

Building upon the taxonomy established by Xia et al. (2024), we organize speculative decoding methods into three primary categories based on their drafting strategies, as illustrated in Figure 1. We treat retrieval-based methods as a distinct category due to their fundamentally different computational characteristics—they employ deterministic lookup operations rather than parametric model inference.

2.1 Independent Drafting

Independent drafting methods employ external models separate from the target LLM for token speculation. These methods either train specialized draft models (Xia et al., 2022; Miao et al., 2023; Zhou et al., 2023) or employ existing small language models from the same family as drafters (Leviathan et al., 2022; Chen et al., 2023; Spector and Re, 2023; Sun et al., 2023).

2.2 Self-Drafting

Self-Drafting techniques leverage components of the target model itself for draft generation, eliminating the need for external models. This category encompasses three distinct approaches.

FFN Heads techniques extend the target model with specialized prediction layers for draft generation (Stern et al., 2018; Cai et al., 2024; Li et al., 2024b,a, 2025). EAGLE (Li et al., 2024b) introduces feature-level autoregressive drafting, where a lightweight transformer layer generates draft tokens based on target model features and arranges them into a token tree for parallel verification. EAGLE-2 (Li et al., 2024a) improves tree construction through dynamic adaptation based on draft model confidence scores, while EAGLE-3 (Li et al., 2025) performs direct token prediction with multi-layer feature fusion, enabling better scaling with increased training data.

Early Exiting methods generate draft tokens using only the initial layers of the target model (Yang et al., 2023b; Zhang et al., 2023; Hooper et al., 2025; Liu et al., 2024). These approaches are based on the observation that language model representations often converge to correct predictions in early layers for simple tokens, making deeper computation unnecessary. Kangaroo (Liu et al., 2024) implements double early exiting by using a shallow sub-network composed of the first few layers of the

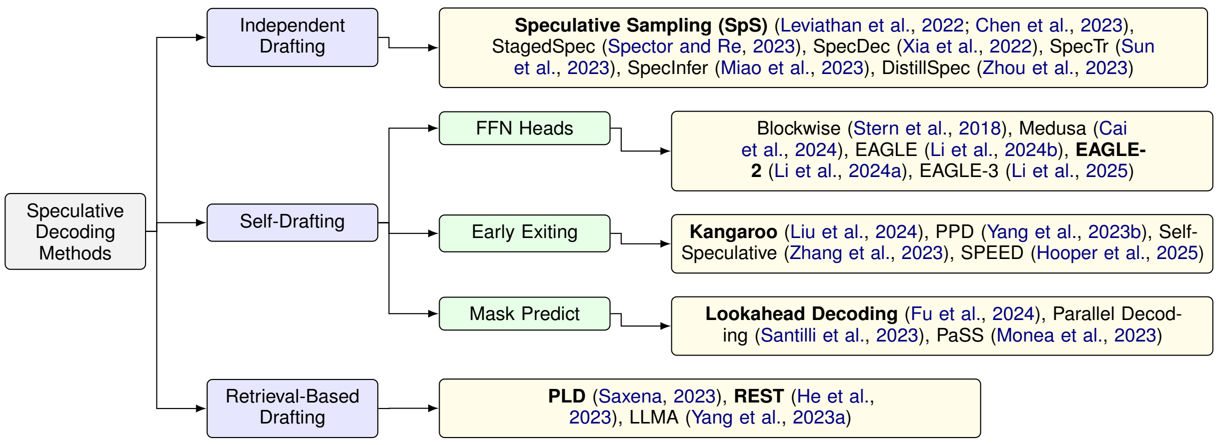
Speculative Decoding Methods
Independent Drafting
Self-Drafting
Retrieval-Based Drafting
FFN Heads
Early Exiting
Mask Predict
Speculative Sampling (SpS) (Leviathan et al., 2022; Chen et al., 2023), StagedSpec (Spector and Re, 2023), SpecDec (Xia et al., 2022), SpecTr (Sun et al., 2023), SpecInfer (Miao et al., 2023), DistillSpec (Zhou et al., 2023)
Blockwise (Stern et al., 2018), Medusa (Cai et al., 2024), EAGLE (Li et al., 2024b), EAGLE-2 (Li et al., 2024a), EAGLE-3 (Li et al., 2025)
Kangaroo (Liu et al., 2024), PPD (Yang et al., 2023b), Self-Speculative (Zhang et al., 2023), SPEED (Hooper et al., 2025)
Lookahead Decoding (Fu et al., 2024), Parallel Decoding (Santilli et al., 2023), PaSS (Monea et al., 2023)
PLD (Saxena, 2023), REST (He et al., 2023), LLMA (Yang et al., 2023a)

Figure 1: Taxonomy of speculative decoding methods categorized by drafting strategy. Methods evaluated in our study are shown in bold.

target model followed by a trained adapter and by stopping the drafting phase when confidence falls below a threshold for each generated token.

Mask Predict strategies employ non-autoregressive mechanisms for parallel token prediction (Santilli et al., 2023; Fu et al., 2024; Monea et al., 2023). Lookahead Decoding (Fu et al., 2024) reformulates autoregressive decoding as Jacobi iteration and employs an n-gram pool that caches token sequences, enabling parallel generation of multiple tokens and accelerating future predictions when similar patterns recur.

2.3 Retrieval-Based Drafting

Retrieval-based methods employ lookup mechanisms rather than model inference for draft generation (Saxena, 2023; He et al., 2023; Yang et al., 2023a). Prompt Lookup Decoding (PLD) (Saxena, 2023) matches n-grams within the generation context itself, using subsequent tokens from context matches as draft candidates. REST (He et al., 2023) extends the retrieval concept by constructing external datastores from large text corpora. During inference, the system retrieves relevant text spans that share common prefixes with the current generation, using the continuations as draft candidates. The retrieved candidates are organized into tree structures for efficient parallel verification.

3 Methodology

We evaluate speculative decoding methods from all three primary categories of the taxonomy (Figure 1), examining their effectiveness across representative small models using established benchmarks and evaluation protocols.

3.1 Models and Datasets

We select three small language models as targets: Qwen2.5-1.5B-Instruct (Yang et al., 2024a), SmolLM2-1.7B-Instruct (Allal et al., 2025), and Llama-3.2-1B-Instruct. For comparative context with existing literature, we include Vicuna-7B-v1.3 (Zheng et al., 2023) as a reference baseline.

For independent drafting methods, we use family-matched external drafters: Qwen2.5-0.5B-Instruct as a drafter for Qwen2.5-1.5B-Instruct, SmolLM2-135M-Instruct as a drafter for SmolLM2-1.7B-Instruct, and Vicuna-68m (Yang et al., 2024b) for Vicuna-7B-v1.3. These represent the smallest available instruction-tuned models from their respective families. Llama-3.2-1B-Instruct was excluded from independent drafting experiments due to the absence of smaller models within the Llama-3.2 family.

We employ SpecBench (Xia et al., 2024) as our evaluation framework, which encompasses six tasks: multi-turn conversation, summarization, retrieval-augmented generation, translation, question answering, and mathematical reasoning. Each task contains 80 instances, totaling 480 evaluation samples. We extend the original framework, which supports only Vicuna models, by integrating Kangaroo and adapting all evaluated methods to support our target models (Qwen2.5-1.5B, SmolLM2-1.7B, Llama-3.2-1B), and by adding fine-grained timing instrumentation for component-level analysis.

3.2 Evaluated Methods

Our evaluation covers all three primary categories from the established taxonomy (Figure 1), with specific representatives for each subcategory.

Independent Drafting is represented by Speculative Sampling (SpS) (Leviathan et al., 2022; Chen et al., 2023) with the family-matched external drafters described above.

Self-Drafting methods are evaluated through three subcategories: EAGLE-2 (Li et al., 2024a) for FFN Heads, Kangaroo (Liu et al., 2024) for Early Exiting, and Lookahead Decoding (Fu et al., 2024) for Mask Predict. We evaluate EAGLE-2 as the publicly available implementation of EAGLE-3 (Li et al., 2025) was still under active development at the time of our experiments.

Retrieval-Based methods include PLD (Saxena, 2023) and REST (He et al., 2023).

For the two methods requiring training (Kangaroo and EAGLE-2), we train model-specific components using the ShareGPT dataset (Aeala, 2023) following the training procedures and hyperparameters specified in the original papers (Liu et al., 2024; Li et al., 2024a). Training is conducted on a single NVIDIA A5500 GPU. For REST, we construct two datastores of different sizes: a large datastore using UltraChat (Ding et al., 2023) and a small datastore using the ShareGPT dataset (Aeala, 2023), following the published methodology (He et al., 2023). We evaluate REST with both datastore configurations.

3.3 Experimental Configuration

All experiments use greedy sampling ($T = 0$), batch size of 1, and float16 precision, following standard protocols established in speculative decoding evaluation (Xia et al., 2024). Experiments are conducted on a single NVIDIA A5500 GPU with 24GB memory.

Each method undergoes systematic hyperparameter optimization to identify optimal configurations for fair comparison. For Vicuna-7B, we use default hyperparameters from SpecBench (Xia et al., 2024) to maintain consistency with established benchmarks. Detailed hyperparameter optimization results are provided in Appendix C.

We report mean performance across three independent runs.

3.4 Evaluation Metrics

We measure two primary metrics that capture different aspects of speculative decoding performance.

Speedup Ratio (Equation 1) compares wall-clock inference time of speculative decoding against standard autoregressive generation, directly measuring the actual acceleration achieved.

Mean Generated Length (MGL) measures the average number of tokens generated per speculative decoding iteration:

$$\text{MGL} = 1 + \mathbb{E}[\text{\# accepted tokens}] \qquad (2)$$

MGL quantifies draft quality and corresponds to the numerator in Equation 1. It represents the upper bound for speedup when drafting costs are negligible (He et al., 2023). Together, these metrics reveal the fundamental trade-off for small model acceleration: MGL captures draft quality, while the gap between MGL and actual speedup exposes drafting overhead (C_d/C_v ratio).

All speculative decoding methods maintain exact output equivalence with standard autoregressive decoding. We therefore do not include output quality measures in our evaluation.

4 Results

4.1 Independent Drafting Evaluation

Speculative Sampling, successful for large models, fails to accelerate small models due to drafting overhead. Table 1 shows that while Speculative Sampling achieves 1.69× average speedup for Vicuna-7B, it experiences severe slowdowns for small models (0.83× for Qwen2.5-1.5B, 0.67× for SmolLM2-1.7B). The timing breakdown in Figure 2 reveals the underlying cause: to generate only two draft tokens, Qwen2.5-0.5B-Instruct requires a drafting time of 16.6ms, while verification time is 10.7ms.

This timing pattern has profound implications for acceleration potential. Since Speedup $= \frac{\text{MGL}}{1+C_d/C_v}$ (Equation 1), achieving unit speedup (Speedup $= 1.0$) requires MGL $= 1 + \frac{C_d}{C_v} = 1 + \frac{16.6}{10.7} = 2.55$. Given that MGL $= 1 + \mathbb{E}[\text{\# accepted tokens}]$ (Equation 2), this translates to requiring 1.55 out of 2 draft tokens (77.5%) to be accepted—a difficult acceptance rate to achieve. The situation becomes even more challenging for SmolLM2-1.7B, where achieving unit speedup requires an acceptance rate of 94%.

Method	Category	Qwen2.5-1.5B		SmolLM2-1.7B		Llama-3.2-1B		Vicuna-7B	
		Speedup	MGL	Speedup	MGL	Speedup	MGL	Speedup	MGL
SpS	Independent Drafting	0.83 ±0.02	2.39	0.67 ±0.00	2.33	N/A	N/A	1.69* ±0.00	2.82
Lookahead	Self-Drafting	1.28 ±0.01	1.85	1.15 ±0.00	1.80	1.21 ±0.00	1.99	1.26 ±0.00	1.75
REST (small)	Retrieval-Based	1.19 ±0.01	1.41	1.17 ±0.00	1.55	1.01 ±0.01	1.42	1.35 ±0.00	1.62
REST (large)	Retrieval-Based	1.08 ±0.02	1.55	1.10 ±0.01	1.69	0.92 ±0.00	1.59	1.43 ±0.01	1.87
PLD	Retrieval-Based	1.52* ±0.02	1.70	1.34* ±0.00	1.66	1.25* ±0.00	1.54	1.61 ±0.00	1.81
Kangaroo	Self-Drafting	1.15 ±0.03	1.93	1.16 ±0.00	1.75	0.91 ±0.01	1.50	1.40 ±0.00	2.22
EAGLE-2	Self-Drafting	**1.70** ±0.00	3.02	**1.81** ±0.01	3.54	**1.44** ±0.00	3.06	**2.32** ±0.00	4.25

Table 1: Performance comparison of speculative decoding methods across evaluated language models. Methods are organized by training requirements (above line: training-free; below line: requires training). Speedup ratios and Mean Generated Length (MGL) are averaged across all SpecBench tasks. Standard deviations over three independent runs are reported for speedup ratios; MGL values are identical across runs due to fixed random seeds. Bold indicates best overall performance per model; asterisks mark best training-free performance.

To investigate the cause of elevated drafter overhead, we analyzed the architectural characteristics that determine inference speed for both target models and drafters in standard autoregressive generation. Results are presented in Table 2. We observe that throughput inversely correlates strongly with model depth rather than parameter count. For instance, the Vicuna-68M drafter (only 2 layers deep) achieves 12× higher throughput than SmolLM2-135M (30 layers), despite having half the parameters. This finding aligns with recent work that demonstrated through extensive experiments that wide-shallow models enable effective drafting while deep-narrow designs create latency bottlenecks (Yan et al., 2024).

Model	Parameters	Layers	Tokens/ Second
Vicuna-7B-v1.3	7B	32	18
Vicuna-68M	68M	2	1259
Qwen2.5-1.5B	1.5B	28	66
Qwen2.5-0.5B	0.5B	24	120
SmolLM2-1.7B	1.7B	24	64
SmolLM2-135M	135M	30	99
Llama-3.2-1B	1B	16	90

Table 2: Model architecture specifications and inference throughput comparison. Inference speed measured for 100-token sequence generation with 100-token input prompts on a single NVIDIA A5500 GPU.

This reveals a fundamental challenge in applying independent drafting to small model acceleration: not only is the availability of smaller models from the same family limited, but the available candidates typically have deep-narrow architectures optimized for standalone language modeling performance, creating high inference latency that renders them unsuitable for speculative decoding applications.

Training specialized shallow drafters from scratch could theoretically address this architectural mismatch, though the significant resource requirements and uncertain performance outcomes make this approach impractical for immediate deployment while remaining an interesting direction for future research.

Key Insight 1

Architecture Determines Viability: When target models are small, traditional independent drafting fails due to the gap between available deep-narrow and optimal wide-shallow drafter architectures. As shown in Table 2, model depth, not parameter count, controls inference speed—making the architectural compatibility the primary constraint for drafting success. Figure 2 demonstrates how this creates prohibitive drafting overhead for small models.

4.2 Self-Drafting Methods

4.2.1 Kangaroo Evaluation

Kangaroo, which employs early exit-based drafting through shallow sub-networks, demonstrates modest improvements compared to independent drafting but fails to achieve meaningful acceleration, with average speedup ratios of 1.15× for Qwen2.5-1.5B, 1.16× for SmolLM2-1.7B, and 0.91× for Llama-3.2-1B across SpecBench tasks (Table 1).

The core limitation stems again from drafting overhead (Figure 2) that is not compensated by sufficient draft quality, as evidenced by modest MGL values of 1.50-1.93 in Table 1.

To understand the sources of this overhead, we analyzed the timing components of individual drafting steps, as illustrated in Figure 3. Despite using

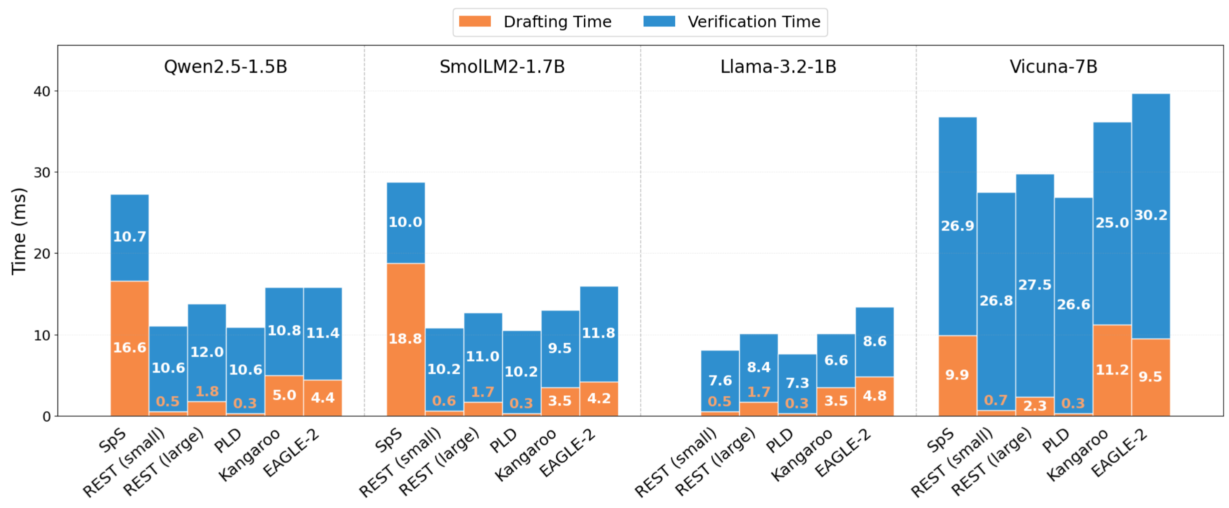

Figure 2: Timing breakdown of drafting and verification phases across methods and models. Drafting time (orange) represents the total time to generate the complete draft sequence (length varies by method); verification time (blue) covers target model forward pass and token acceptance. Values averaged over three independent runs. Lookahead Decoding excluded due to non-sequential draft-verify structure; SpS not available for Llama-3.2-1B.

only two layers for drafting, the shared language modeling head creates a substantial bottleneck, consuming 17.3% to 33% of total drafting time depending on vocabulary size. This results in total drafting step times of 2.2ms for Qwen2.5-1.5B and Llama-3.2-1B (models with larger vocabularies), approaching Vicuna-7B's 2.5ms despite their verification times being 2.3× and 3.8× shorter respectively (Figure 2), making the drafting overhead particularly significant for small models. Comprehensive timing analysis and component breakdowns for all methods are provided in Appendix D.

Additionally, Llama-3.2-1B faces a specific architectural disadvantage: with only 16 layers total, using 2 layers for drafting (12.5% of the model) provides minimal computational savings compared to deeper models where 2 layers represent only 6.7-8.3% of the total architecture, making early exiting less beneficial and resulting in slowdown rather than speedup (Table 1).

4.2.2 EAGLE-2 Evaluation

EAGLE-2 demonstrates superior performance among all methods (Table 1), achieving 1.44-1.81x speedup across small models and tasks through higher draft quality, attaining the highest MGL values across all evaluated methods.

This success stems from two key technical features: feature-level autoregressive drafting using a specialized draft model (a single transformer layer) trained specifically for speculation, and token tree verification that maximizes acceptance probability

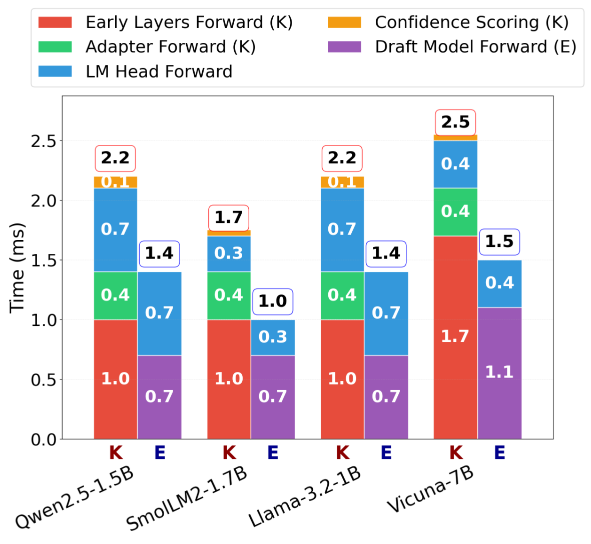

Figure 3: Timing breakdown comparison between Kangaroo and EAGLE-2 autoregressive drafting methods across all evaluated language models. Each bar shows the time required for individual components during a single forward pass of the drafter that generates one draft token. K denotes Kangaroo and E denotes EAGLE. Values are averaged over three independent runs.

through parallel candidate evaluation.

The component breakdown in Figure 3 reveals that while the LM Head still represents a significant portion of drafting time, EAGLE-2's use of only a trained transformer layer for drafting, compared to Kangaroo's combination of target model layers plus trained adapter, makes it significantly faster, achieving 1.0-1.4ms total drafting step time compared to Kangaroo's 1.7-2.2ms. This demonstrates that EAGLE-2 succeeds in creating not only a highly accurate drafter (high MGL) but also an efficient one, thus achieving the high speedups observed in our evaluation.

Hyperparameter tuning (see Appendix C) revealed that small models favor shallow, wide tree configurations (depth of 2-3, width of 20) rather than the deep trees optimal for Vicuna-7B (depth of 5, width of 10). Since each tree depth level requires an additional forward pass of the draft model, shallow configurations reduce the computational cost of candidate generation, showing that even for EAGLE-2, the best trade-off is achieved by minimizing drafting overhead through reduced forward passes.

Key Insight 2

Model-Specific Training of Efficient Drafting Modules Can Justify Drafting Overhead: EAGLE-2's success demonstrates that when drafters are specifically trained and efficiently designed, the resulting draft quality can be high enough to overcome the drafting overhead (Table 1). Using only a trained transformer layer that efficiently reuses computations already performed by the target model for drafting enables both high accuracy and computational efficiency (Figure 3), while Kangaroo's approach of combining target model layers with a trained adapter creates bottlenecks that prevent effective acceleration despite model-specific training.

4.2.3 Lookahead Decoding Evaluation

Lookahead Decoding achieved modest but consistent average speedup of 1.28× for Qwen2.5-1.5B, 1.15× for SmolLM2-1.7B, and 1.21× for Llama-3.2-1B across SpecBench tasks (Table 1). The method's performance remained relatively stable across model sizes, with Vicuna-7B achieving comparable 1.26× speedup.

Unlike traditional speculative decoding methods that use external draft models, Lookahead employs a reformulation of autoregressive decoding as Jacobi iteration to generate multiple n-grams in parallel within the target model itself. The method maintains a fixed-size 2D window characterized by W (future token positions for parallel speculation), N (lookback steps for n-gram collection), and G (n-gram candidates verified in parallel). By increasing these parameters, the method trades higher per-step computation for reduced total decoding steps, achieving higher Mean Generated Length (MGL).

The original Lookahead Decoding paper reports that smaller models achieve higher speedups because they operate further from GPU computational limits, enabling larger W, N, and G configurations. Our results confirm this pattern: hyperparameter optimization revealed substantially larger optimal window sizes for small models compared to Vicuna-7B (see Appendix C), which translate to higher MGL values: 1.80-1.99 across small models versus 1.75 for Vicuna-7B.

Model	**Time [ms]**		**Overhead**
	Lookahead	Autoregressive	**Ratio**
Llama-3.2-1B	9.4	6.4	1.47×
Qwen2.5-1.5B	13.1	9.9	1.33×
SmolLM2-1.7B	13.2	8.9	1.47×
Vicuna-7B	31.6	25.8	1.23×

Table 3: Per-iteration execution time comparison between Lookahead Decoding and standard autoregressive generation. Overhead ratio indicates the relative cost of Lookahead's parallel speculation mechanism. Values are averaged over three independent runs on SpecBench dataset.

However, our timing analysis reveals a critical counterbalancing factor. Table 3 shows that per-iteration overhead is proportionally more significant for small models (33-47% increase) than for Vicuna-7B (23% increase). We hypothesize that this overhead stems from fixed computational costs in operations such as n-gram pool management and attention mask construction. When baseline iteration times are 6-10ms for small models versus 26ms for Vicuna-7B, these fixed costs become proportionally more expensive, offsetting the computational advantages and resulting in similar speedups across model scales.

4.3 Retrieval-Based Methods

4.3.1 PLD Evaluation

Prompt Lookup Decoding achieves acceleration across all evaluated small models (1.25-1.52×) with negligible 0.3ms overhead (Figure 2), enabling effective speculation despite modest drafting quality (Table 1). Among all training-free methods evaluated, PLD consistently emerges as the best performer, achieving the highest average speedups across tasks for all small models while maintaining universal applicability without requiring model-specific training or external resources.

However, task-specific analysis (Table 4) reveals that PLD's effectiveness varies significantly across different scenarios, showing systematic patterns: thanks to context reusability, the highest speedups are achieved in input-grounded tasks like summarization (1.28-2.52×) and RAG (1.51-1.60x), in which PLD competes and in some cases even surpasses EAGLE-2, while translation (1.03-1.28x) and short question answering (1.02-1.10x) show limited gains due to the context containing minimal information useful for generation.

4.3.2 REST Evaluation

REST evaluation reveals that retrieval overhead has a greater impact on small models than large ones. Our performance and timing analysis (Table 1 and Figure 2) demonstrates this disparity: while Vicuna-7B tolerates 2.3ms external retrieval overhead and actually benefits from larger datastores (achieving 1.43× vs 1.35× speedup), small models' reduced verification times (7.3-12.0ms) make any additional latency prohibitively expensive. Consequently, large datastore configurations achieve only 0.92-1.10× average speedup for small models, with Llama-3.2-1B experiencing slowdown. Additionally, deploying large datastores (∼12GB in our evaluation) presents significant storage challenges for edge devices with limited memory capacity, making such configurations impractical for resource-constrained environments where small models are typically deployed. Using small datastores reduces retrieval cost to negligible levels similar to PLD (0.5-0.7ms), but the limited datastore size results in fewer matches and lower MGL values, ultimately yielding performance still inferior to PLD (1.01-1.19× vs PLD's 1.25-1.52×).

Our evaluation uses general-purpose datastores across all tasks, but task-specific datastores could potentially improve retrieval quality and MGL values without requiring larger datastores. However, we chose to use general-purpose datastores to ensure fair comparison across all methods, following the established evaluation protocol used in SpecBench (Xia et al., 2024).

Key Insight 3

Negligible-Cost Drafting Enables Consistent Gains: When verification times are short, drafting overhead becomes a critical bottleneck. As shown in Figure 2, methods that minimize computational overhead during speculation (like PLD) provide a reliable path to acceleration, as evidenced by the consistent speedups (Table 1).

4.4 Tree Verification Impact

We conduct an ablation study examining EAGLE-2's performance with and without tree verification, which determines whether the method verifies multiple candidate sequences in parallel or processes a single candidate sequence.

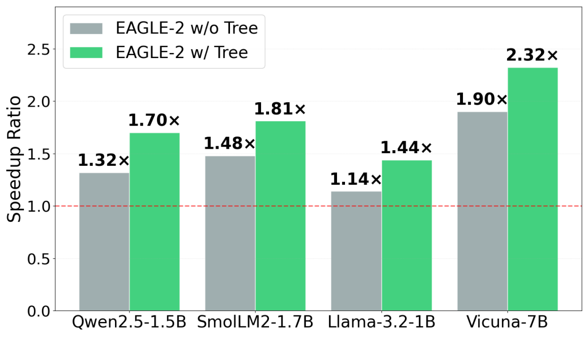

Figure 4: Ablation study comparing EAGLE-2 performance with and without tree verification across all evaluated models. Speedup ratios show average performance across SpecBench tasks.

Our ablation study reveals consistent 22-29% performance improvements from tree verification across all small models (Figure 4), aligning with prior findings on large models (Li et al., 2024b). Importantly, we observe no additional computational overhead in the verification phase when adding the tree component. These results demonstrate that tree verification constitutes an acceleration booster independent of model scale, suggesting that it could be integrated with other drafting techniques to enhance their overall effectiveness.

Key Insight 4

Tree Verification Boosts Acceleration: Tree-based verification consistently improves EAGLE-2 performance regardless of model size (Figure 4), suggesting it may be a valuable component for methods that can incorporate tree structures.

5 Conclusion

In this work, we conduct a comprehensive study of speculative decoding techniques for small language models (1-2B parameters), revealing that effective acceleration in this regime requires different approaches compared to those optimized for larger models. Through systematic evaluation across five method categories, we demonstrate that drafting overhead becomes the primary bottleneck for small model acceleration, fundamentally altering the cost-benefit calculus that drives method selection. To our knowledge, this is the first empirical analysis dedicated to speculative decoding for small language models. Our aim is to provide insights for future research and practical guidance for small model acceleration.

Limitations

This paper provides a comprehensive benchmarking of speculative decoding methods across multiple categories to establish their effectiveness for small language models. However, several limitations should be acknowledged.

Our evaluation of independent drafting methods relies exclusively on existing family-matched models, which limits the scope of our conclusions for this category. We did not evaluate independent drafting with purpose-built draft models that could be fine-tuned or trained from scratch specifically for speculative decoding tasks. Future work with custom draft models optimized for efficient speculation rather than standalone language modeling performance could potentially overcome the architectural incompatibilities and overhead limitations we identified for independent drafting approaches.

The experimental analysis is conducted entirely on a single GPU architecture (NVIDIA A5500) with specific memory bandwidth and computational characteristics. Different hardware configurations, including other GPU families or edge computing devices, might produce different overhead-performance trade-offs, potentially altering the relative rankings of methods.

Ethical Considerations

All datasets and scientific artifacts used in our experiments are available for research use under permissive licenses, and their use in this paper is consistent with their intended purposes. Our research focuses on inference efficiency optimization for resource-constrained environments, which could potentially democratize access to language model capabilities by reducing computational requirements. The acceleration techniques studied preserve exact output equivalence with standard generation methods, ensuring no degradation in model behavior or introduction of unintended biases. We acknowledge that more efficient inference could enable broader deployment of language models, including potential misuse scenarios. However, our work does not modify model capabilities or outputs, only the speed of generation. The responsibility for appropriate use remains with downstream deployers, consistent with the original model licenses and intended use policies.

Acknowledgments

This work was supported by European Union's Horizon Europe research and innovation programme under grant agreement number 101214398 (ELLIOT).

References

Aeala. 2023. Sharegpt v4.3 unfiltered cleaned split.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, and 1 others. 2025. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, De huai Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *ArXiv*, abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, L. Sifre, and John M. Jumper. 2023. Accelerating large language model decoding with speculative sampling. *ArXiv*, abs/2302.01318.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*.

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

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Hasan Genc, Kurt Keutzer, Amir Gholami, and Yakun Sophia Shao. 2025. Speed: Speculative pipelined execution for efficient decoding. In *Enhancing LLM Performance: Efficacy, Fine-Tuning, and Inference Techniques*, pages 19–32. Springer.

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.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2022. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024a. Eagle-2: Faster inference of language models with dynamic draft trees. In *Conference on Empirical Methods in Natural Language Processing*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024b. Eagle: Speculative sampling requires rethinking feature uncertainty. *ArXiv*, abs/2401.15077.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024. Kangaroo: Lossless self-speculative decoding via double early exiting. *ArXiv*, abs/2404.18911.

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. 2023. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*.

Giovanni Monea, Armand Joulin, and Edouard Grave. 2023. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*.

Apoorv Saxena. 2023. Prompt lookup decoding.

Noam M. Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *ArXiv*, abs/1911.02150.

Benjamin Spector and Christal Re. 2023. Accelerating llm inference with staged speculative decoding. *ArXiv*, abs/2308.04623.

Mitchell Stern, Noam M. Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Neural Information Processing Systems*.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2023. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36:30222–30242.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2022. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Conference on Empirical Methods in Natural Language Processing*.

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. *ArXiv*, abs/2401.07851.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2024. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*.

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

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, and 25 others. 2024a. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024b. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2023b. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. *ArXiv*, abs/2307.05908.

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

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *ArXiv*, abs/2401.02385.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*.

A Background

This section establishes the formal framework for speculative decoding, covering notation, autoregressive decoding, and the speculative decoding paradigm.

A.1 Notation

Let $\mathcal{V}$ denote the vocabulary of discrete tokens. We represent a token sequence as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ where each $x_i \in \mathcal{V}$. For notational convenience, we denote the prefix sequence up to and including position t as $x_{\leq t} = (x_1, \ldots, x_t)$.

A language model $M : \mathcal{V}^* \to \Delta^{|\mathcal{V}|-1}$ maps a sequence of tokens to a probability distribution over the vocabulary, where $\mathcal{V}^*$ denotes the set of all possible finite sequences over vocabulary $\mathcal{V}$, and $\Delta^{|\mathcal{V}|-1}$ denotes the probability simplex over $\mathcal{V}$. We distinguish between the target model M_q that we aim to accelerate, and a draft model M_p used for speculation.

We write $x \sim p$ to denote that x is sampled from distribution p.

A.2 Autoregressive Decoding

Transformer-based language models generate text through autoregressive decoding, where tokens are produced sequentially, each conditioned on all previous tokens. Given an initial context $\mathbf{x}^{(0)} = (x_1, \ldots, x_t)$, the model generates the next token by first computing the conditional probability distribution:

$$q(x_{t+1}|x_{\leq t}) = M_q(\mathbf{x}^{(0)}) \tag{3}$$

where $q(x_{t+1}|x_{\leq t})$ represents the probability distribution computed by the target model M_q over the vocabulary. The actual token x_{t+1} is then obtained by sampling from this distribution using various strategies such as greedy sampling (i.e., applying an argmax function on the distribution), top-k sampling, or nucleus (top-p) sampling (Kool et al., 2020; Holtzman et al., 2019).

This process continues autoregressively. The newly generated token is appended to the sequence:

$$\mathbf{x}^{(1)} = (\mathbf{x}^{(0)}, x_{t+1}) = (x_1, \ldots, x_t, x_{t+1}) \tag{4}$$

and becomes part of the input for the next iteration. The autoregressive chain continues as:

$$q(x_{t+2}|x_{\leq t+1}) = M_q(\mathbf{x}^{(1)}) \tag{5}$$
$$x_{t+2} \sim q(x_{t+2}|x_{\leq t+1}) \tag{6}$$
$$\mathbf{x}^{(2)} = (\mathbf{x}^{(1)}, x_{t+2}) \tag{7}$$

until a termination condition is met (e.g., generating an end-of-sequence token or reaching maximum length).

A.3 Speculative Decoding

Speculative decoding adopts a draft-then-verify paradigm where, at each decoding step, multiple draft tokens are generated as speculation for future positions and then verified in parallel using the target model.

A.3.1 Drafting Phase

Given a prefix sequence $x_{\leq t}$, the drafting phase generates K candidate tokens $\tilde{x}_{t+1}, \tilde{x}_{t+2}, \ldots, \tilde{x}_{t+K}$. The procedure can be formalized as:

$$\tilde{x}_{t+1}, \tilde{x}_{t+2}, \ldots, \tilde{x}_{t+K} = \text{DRAFT}(x_{\leq t}, M_p) \tag{8}$$

where DRAFT$(\cdot)$ represents the drafting strategy and M_p denotes the draft model.

The DRAFT$(\cdot)$ strategy can be autoregressive, where tokens are generated sequentially and each token depends on previously drafted tokens:

$$\begin{aligned}\tilde{x}_{t+i} &\sim p(x_{t+i}|x_{\leq t}, \tilde{x}_{t+1}, \ldots, \tilde{x}_{t+i-1}) = \\ &= M_p(x_{\leq t}, \tilde{x}_{t+1}, \ldots, \tilde{x}_{t+i-1}) \\ &\text{for } i = 1, \ldots, K \end{aligned} \tag{9}$$

Alternatively, DRAFT$(\cdot)$ can generate tokens in parallel using non-autoregressive models:

$$\begin{aligned}(\tilde{x}_{t+1}, \ldots, \tilde{x}_{t+K}) &\sim p(x_{t+1}, \ldots, x_{t+K}|x_{\leq t}) = \\ &= M_p(x_{\leq t}) \end{aligned} \tag{10}$$

Finally, some methods (usually retrieval-based methods) bypass parametric models entirely, using a deterministic algorithm A that replaces M_p:

$$(\tilde{x}_{t+1}, \ldots, \tilde{x}_{t+K}) = A(x_{\leq t}, \mathcal{D}) \tag{11}$$

where $\mathcal{D}$ represents auxiliary data such as generation context or external datastores.

A.3.2 Verification Phase

In the verification phase, the target model verifies all drafted tokens in parallel by computing $K + 1$ probability distributions simultaneously:

$$\begin{aligned}q(x_{t+i}|x_{\leq t}, \tilde{x}_{t+1}, \ldots, \tilde{x}_{t+i-1}) &= \\ = M_q(x_{\leq t}, \tilde{x}_{t+1}, \ldots, \tilde{x}_{t+i-1}),& \\ i = 1, \ldots, K+1 & \end{aligned} \tag{12}$$

For notational convenience, we use the abbreviated form q_{t+i} to denote these probability distributions in the following discussion. Each drafted token $\tilde{x}_{t+i}$ is then verified according to a criterion (e.g., greedy verification, speculative sampling, approximate verification). For example, using greedy verification, we accept token $\tilde{x}_{t+i}$ if $\tilde{x}_{t+i} = \arg\max q_{t+i}$, otherwise we reject it.

When a drafted token $\tilde{x}_{t+c}$ fails verification, all subsequent drafted tokens are discarded, and the token at position $t + c$ is corrected as $x_{t+c} = \arg\max q_{t+c}$.

In this way, a speculative decoding iteration generates c tokens using only a single forward pass of the target model (for verification). If all drafted tokens are rejected ($c = 1$), only a single token is generated (the corrected token from the target model). Conversely, if all K drafted tokens pass verification, $K+1$ tokens are generated, as an additional token $x_{t+K+1} = \arg\max q_{t+K+1}$ is added.

B Task-Specific Results

Table 4 presents detailed performance breakdowns across all six SpecBench tasks, showing speedup ratios and Mean Generated Length values for each evaluated method and model combination.

C Hyperparameter Optimization

This section outlines the role of the key hyperparameters used for each method. When not specified by the original implementation, values were tuned using Optuna (Akiba et al., 2019), with the objective of maximizing speedup.

Table 5 summarizes the final values used in our experiments. Descriptions of each hyperparameter and its function are provided below.

- **Speculative Sampling:** *Draft-Length* denotes the number of tokens generated by the draft model before verification.

- **Lookahead Decoding:** *Window* (W) defines the number of future positions speculated in parallel per step; *Level* (N) controls how many past steps (lookback) are used to generate each n-gram; *Guess* (G) limits the number of n-grams verified in parallel to reduce computational cost.

- **REST:** *Num-Draft* is the number of candidate nodes retained in the continuation tree after pruning; *Max-Token-Span* is the maximum suffix length that must match during retrieval.

- **PLD:** *Matching-Window-Size* specifies the maximum n-gram length for matching context; *draft-length* controls the length of the speculative continuation generated by the method.

- **Kangaroo:** *Exit-Layer* determines how shallow the self-draft model is (i.e., how many layers are used for drafting); *Threshold* sets the minimum token-level confidence required for early exit; *Step* defines the maximum number of draft tokens generated per speculative iteration.

- **EAGLE-2:** *Total-Token* is the maximum number of speculative tokens per draft attempt; *Depth* is the number of levels in the token tree; *Top-k* specifies the number of nodes selected from each layer for expansion to the next layer (i.e., the tree width at each level).

Task	Method	Qwen2.5-1.5B		SmolLM2-1.7B		Llama-3.2-1B		Vicuna-7B	
		Speedup	MGL	Speedup	MGL	Speedup	MGL	Speedup	MGL
Multi-Turn Conversation	SpS	0.83 ± 0.02	2.38	0.72 ± 0.00	2.42	N/A	N/A	1.81* ± 0.00	2.74
	Lookahead	1.34 ± 0.02	1.89	1.22 ± 0.00	1.87	1.35* ± 0.01	2.02	1.33 ± 0.00	1.74
	REST (small)	1.27 ± 0.02	1.48	1.33 ± 0.00	1.70	1.13 ± 0.01	1.51	1.54 ± 0.01	1.79
	REST (large)	1.14 ± 0.01	1.63	1.21 ± 0.00	1.83	1.01 ± 0.00	1.67	1.60 ± 0.00	2.02
	PLD	1.46* ± 0.02	1.69	1.49* ± 0.00	1.69	1.31 ± 0.01	1.55	1.56 ± 0.00	1.64
	Kangaroo	1.22 ± 0.05	1.99	1.26 ± 0.01	1.89	0.97 ± 0.00	1.55	1.55 ± 0.00	2.31
	EAGLE-2	**1.78** ± 0.00	3.07	**2.05** ± 0.01	3.81	**1.55** ± 0.00	3.19	**2.75** ± 0.00	4.67
Translation	SpS	0.83 ± 0.02	2.12	0.55 ± 0.00	1.74	N/A	N/A	1.22 ± 0.00	2.68
	Lookahead	1.23* ± 0.01	1.45	1.08 ± 0.00	1.42	1.04 ± 0.00	1.71	1.15 ± 0.00	1.86
	REST (small)	1.16 ± 0.02	1.26	1.06 ± 0.00	1.28	0.90 ± 0.01	1.18	1.22 ± 0.00	1.57
	REST (large)	1.01 ± 0.02	1.40	0.93 ± 0.01	1.39	0.82 ± 0.01	1.27	1.26* ± 0.01	1.88
	PLD	1.03 ± 0.02	1.10	1.17* ± 0.00	1.22	1.28* ± 0.01	1.22	1.19 ± 0.00	1.88
	Kangaroo	0.96 ± 0.01	1.35	1.05 ± 0.01	1.34	0.80 ± 0.01	1.17	1.15 ± 0.00	2.09
	EAGLE-2	**1.55** ± 0.00	2.59	**1.62** ± 0.01	2.79	**1.42** ± 0.01	2.83	**1.98** ± 0.00	3.22
Summarization	SpS	0.75 ± 0.01	2.20	0.67 ± 0.00	2.20	N/A	N/A	1.88 ± 0.00	2.83
	Lookahead	1.24 ± 0.01	1.77	1.14 ± 0.00	1.74	1.19 ± 0.00	1.88	1.19 ± 0.00	1.51
	REST (small)	1.02 ± 0.01	1.24	1.07 ± 0.00	1.32	0.98 ± 0.01	1.30	1.24 ± 0.00	1.38
	REST (large)	0.95 ± 0.02	1.43	1.05 ± 0.01	1.56	0.92 ± 0.00	1.52	1.35 ± 0.01	1.65
	PLD	**2.52*** ± 0.01	2.65	1.43* ± 0.00	1.85	1.28* ± 0.00	1.63	**2.20*** ± 0.00	2.51
	Kangaroo	1.22 ± 0.03	1.95	1.04 ± 0.00	1.50	0.86 ± 0.00	1.35	1.36 ± 0.00	1.95
	EAGLE-2	1.56 ± 0.00	2.86	**1.53** ± 0.00	3.07	**1.30** ± 0.00	2.87	2.15 ± 0.00	3.93
Question Answering	SpS	0.80 ± 0.01	2.20	0.63 ± 0.00	2.18	N/A	N/A	1.76* ± 0.00	3.01
	Lookahead	1.19 ± 0.01	1.66	1.05 ± 0.00	1.60	1.14* ± 0.00	1.96	1.25 ± 0.00	1.82
	REST (small)	1.33* ± 0.01	1.45	1.23 ± 0.00	1.54	1.04 ± 0.01	1.40	1.40 ± 0.00	1.59
	REST (large)	1.29 ± 0.02	1.82	1.26* ± 0.02	1.78	1.01 ± 0.01	1.66	1.64 ± 0.02	1.98
	PLD	1.10 ± 0.02	1.38	1.02 ± 0.00	1.31	1.06 ± 0.01	1.48	1.39 ± 0.00	1.73
	Kangaroo	1.10 ± 0.04	1.86	1.13 ± 0.00	1.68	0.90 ± 0.01	1.54	1.40 ± 0.00	2.20
	EAGLE-2	**1.64** ± 0.01	2.76	**1.76** ± 0.01	3.18	**1.41** ± 0.01	2.88	**2.19** ± 0.00	3.53
Mathematical Reasoning	SpS	0.96 ± 0.02	2.77	0.73 ± 0.00	2.47	N/A	N/A	1.70* ± 0.01	2.77
	Lookahead	1.54* ± 0.01	2.07	1.37 ± 0.00	2.06	1.37* ± 0.00	2.14	1.45 ± 0.00	1.95
	REST (small)	1.20 ± 0.01	1.36	1.19 ± 0.00	1.53	1.02 ± 0.01	1.36	1.37 ± 0.00	1.56
	REST (large)	0.97 ± 0.01	1.40	0.97 ± 0.01	1.49	0.83 ± 0.00	1.39	1.27 ± 0.01	1.66
	PLD	1.40 ± 0.02	1.65	1.38* ± 0.00	1.70	1.17 ± 0.01	1.51	1.66 ± 0.00	1.93
	Kangaroo	1.19 ± 0.04	1.93	1.26 ± 0.00	1.86	0.94 ± 0.01	1.53	1.57 ± 0.00	2.42
	EAGLE-2	**1.94** ± 0.01	3.28	**2.19** ± 0.01	3.98	**1.66** ± 0.00	3.35	**2.80** ± 0.00	4.69
RAG	SpS	0.83 ± 0.02	2.26	0.76 ± 0.00	2.35	N/A	N/A	1.80* ± 0.00	3.03
	Lookahead	1.10 ± 0.01	1.60	1.03 ± 0.00	1.55	1.16 ± 0.00	1.98	1.14 ± 0.00	1.54
	REST (small)	1.12 ± 0.01	1.40	1.13 ± 0.00	1.45	0.97 ± 0.00	1.33	1.29 ± 0.00	1.51
	REST (large)	1.10 ± 0.03	1.68	1.18 ± 0.02	1.74	0.94 ± 0.01	1.56	1.45 ± 0.01	1.83
	PLD	1.51* ± 0.02	1.59	1.60* ± 0.00	1.79	**1.52*** ± 0.00	1.62	1.74 ± 0.00	1.89
	Kangaroo	1.20 ± 0.04	1.83	1.18 ± 0.01	1.70	0.97 ± 0.00	1.50	1.36 ± 0.00	2.04
	EAGLE-2	**1.55** ± 0.00	2.92	**1.63** ± 0.01	3.39	1.31 ± 0.00	2.98	**1.91** ± 0.00	3.84

Table 4: Task-specific performance breakdown showing speedup ratios and Mean Generated Length (MGL) for all evaluated methods and models across six SpecBench tasks. Standard deviations over three independent runs are reported for speedup ratios; MGL values are identical across runs due to fixed random seeds. Bold indicates best overall performance per task and model; asterisks mark best training-free performance.

Method	Hyperparameter	Model			
		Qwen2.5-1.5B	SmolLM2-1.7B	Llama-3.2-1B	Vicuna-7B
SpS	Draft-Length	2	2	N/A	10
Lookahead	Level	4	4	6	5
	Window	20	20	10	7
	Guess	20	20	10	7
REST (small)	Num-Draft	64	64	48	64
	Max-Token-Span	4	3	3	16
REST (large)	Num-Draft	64	48	64	64
	Max-Token-Span	5	10	5	16
PLD	Matching-Window-Size	3	3	3	3
	Draft-Length	20	15	15	10
Kangaroo	Exit-Layer	2	2	2	2
	Threshold	0.6	0.7	0.8	0.3
	Step	7	7	4	6
EAGLE-2	Total-Token	48	48	48	60
	Depth	2	3	2	5
	Top-k	20	20	20	10

Table 5: Hyperparameter values used for each method across different models. When not provided by the original implementation, parameters were tuned to maximize speedup.

D Detailed Timing Breakdown

To provide deeper insights into the computational overhead characteristics of different speculative decoding approaches, we present a timing breakdown of the drafting phases for all evaluated methods. All timing measurements are averaged over three independent runs across the complete SpecBench evaluation dataset, following the same experimental protocol used for speedup analysis. We distinguish between autoregressive methods (Speculative Sampling, Kangaroo, and EAGLE-2) and retrieval-based methods (PLD and REST).

D.1 Autoregressive Methods

For autoregressive drafting methods, we measure the time required for individual components within a single drafting step (i.e., the generation of one draft token). Table 6 presents these measurements across all evaluated models.

Speculative Sampling consists solely of the draft model forward pass. The timing difference between small models' drafters (8.3ms and 9.4ms for Qwen2.5-1.5B and SmolLM2-1.7B, respectively) and the Vicuna drafter (1.0ms) confirms the substantial overhead identified in Section 4.1

Kangaroo presents a more complex architecture, decomposing into four components: *Early Layers Forward* (first two layers of the target model), *Adapter Forward* (the trained adapter component), *LM Head Forward* (target model projection to vocabulary space), and *Confidence Scoring* (early stopping computation). The results reveal that drafting overhead for small models (2.2ms for Qwen2.5-1.5B/Llama-3.2-1B, 1.7ms for SmolLM2-1.7B) approaches that of Vicuna-7B (2.5ms). When compared to the verification times shown in Figure 2, this represents a much higher proportional cost for small models, substantially reducing Kangaroo's effectiveness. Additionally, Qwen2.5-1.5B and Llama-3.2-1B exhibit significant LM Head overhead, consuming 32% of total drafting time (0.7ms out of 2.2ms) compared to 18% for SmolLM2-1.7B and 16% for Vicuna, reflecting their larger vocabulary sizes (152K and 128K tokens respectively, versus 49K for SmolLM2 and 32K for Vicuna).

In contrast, EAGLE-2 requires only the trained draft model forward pass and target model LM Head projection. Despite sharing the same LM Head overhead as Kangaroo, EAGLE-2 achieves significantly lower total drafting time (57-70% reduction compared to Kangaroo across models) by using a more efficient draft model architecture—a single transformer layer versus Kangaroo's two layers plus adapter.

D.2 Retrieval-Based Methods

For retrieval-based methods, we measure the entire drafting process that produces complete speculation candidates. This differs from autoregressive methods as retrieval-based drafting generates full

Method	Component	Time [ms]			
		Qwen2.5-1.5B	SmolLM2-1.7B	Llama-3.2-1B	Vicuna-7B
SpS	*Draft Model Forward*	8.3	9.4	N/A	1.0
Kangaroo	*Early Layers Forward*	1.0	1.0	1.0	1.7
	Adapter Forward	0.4	0.4	0.4	0.4
	LM Head Forward	0.7	0.3	0.7	0.4
	Confidence Scoring	0.1	<0.05	0.1	<0.05
	Total	**2.2**	**1.7**	**2.2**	**2.5**
EAGLE	*Draft Model Forward*	0.7	0.7	0.7	1.1
	LM Head Forward	0.7	0.3	0.7	0.4
	Total	**1.4**	**1.0**	**1.4**	**1.5**

Table 6: Timing breakdown for autoregressive drafting methods. Measurements show the time required for each component within a single drafting step that generates one draft token and are averaged over three independent runs.

Method	Component	Time [ms]							
		Qwen2.5-1.5B		SmolLM2-1.7B		Llama-3.2-1B		Vicuna-7B	
		Match	No-match	Match	No-match	Match	No-match	Match	No-match
PLD	*Context Search*	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
REST (Large)	*Datastore Search*	0.4	0.4	0.6	0.6	0.3	0.3	1.1	1.1
	Tree Construction + Pruning	1.2	N/A	0.7	N/A	1.2	N/A	0.7	N/A
	Other Operations	0.5	0.3	0.4	0.3	0.4	0.3	0.5	0.4
	Total	**2.1**	**0.7**	**1.7**	**0.9**	**1.9**	**0.6**	**2.3**	**1.5**
REST (Small)	*Datastore Search*	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.2
	Tree Construction + Pruning	0.3	N/A	0.3	N/A	0.3	N/A	0.2	N/A
	Other Operations	0.2	0.1	0.3	0.1	0.3	0.1	0.3	0.2
	Total	**0.6**	**0.2**	**0.7**	**0.2**	**0.6**	**0.2**	**0.7**	**0.4**

Table 7: Timing breakdown for retrieval-based methods. Measurements distinguish between iterations with context/datastore matches versus no matches found and are averaged over three independent runs.

sequences (PLD) or token trees (REST) rather than individual tokens. We distinguish between cases where context matches are found versus cases with no matches, as this significantly affects the computational requirements.

Table 7 presents the detailed breakdown. For REST, "Other Operations" encompasses all preparation steps required for the verification phase, including tree linearization, attention mask preparation, and KV cache management.

Despite managing a large datastore (∼12GB), REST's search time for small models (0.3-0.6ms) remains comparable to PLD's context search, likely benefiting from REST's optimized multi-threaded Rust implementation with suffix arrays compared to PLD's single-threaded Python-based string matching approach. However, tree construction emerges as the primary bottleneck, consuming 41-63% of total drafting time for small models.

This large overhead can be explained by examining the relationship between the *Max-Token-Span* hyperparameter (see Appendix C) and computational cost. Lower *Max-Token-Span* values find matches with fewer search iterations but typically retrieve more candidate continuations due to the less restrictive matching criteria, while higher values require more iterations (decreasing token-span until a match is found) before finding matches but generally retrieve fewer continuations. Qwen2.5-1.5B and Llama-3.2-1B (*Max-Token-Span*=5) exhibit similar behavior with substantial tree construction costs (1.2ms) due to frequently retrieving many candidate continuations that create large pre-pruning trees. At the opposite extreme, Vicuna-7B (*Max-Token-Span*=16) spends proportionally more time on datastore search (1.1ms) and less on tree management (0.7ms), as its higher initial token-span requires more search iterations when the current iteration fails to find matches, but this configuration tends to result in smaller trees when matches are eventually found. SmolLM2, with its intermediate *Max-Token-Span* value (10 tokens), exhibits behavior between these extremes.