PipeSpec: Breaking Stage Dependencies in Hierarchical LLM Decoding

Bradley McDanel[1] **Sai Qian Zhang**[2] **Yunhai Hu**[2] **Zining Liu**[3]
[1]*Franklin and Marshall College* [2]*New York University* [3]*University of Pennsylvania*
bmcdanel@fandm.edu {yunhai.hu, sai.zhang}@nyu.edu
zliu0@seas.upenn.edu

Abstract

Speculative decoding accelerates large language model inference by using smaller draft models to generate candidate tokens for parallel verification. However, current approaches are limited by sequential stage dependencies that prevent full hardware utilization. We present PipeSpec, a framework that generalizes speculative decoding to use multiple models arranged in a hierarchical pipeline, enabling asynchronous execution with lightweight coordination for prediction verification and rollback. Our analytical model characterizes token generation rates across pipeline stages and proves guaranteed throughput improvements over traditional decoding for any non-zero acceptance rate. We further derive closed-form expressions for steady-state verification probabilities that explain the empirical benefits of pipeline depth. We validate PipeSpec across text summarization, mathematical reasoning, and code generation tasks using LLaMA 2 and 3 models, demonstrating that pipeline efficiency increases with model depth, providing a scalable approach to accelerating LLM inference on multi-device systems. Our code is available at https://github.com/BradMcDanel/PipeSpec.

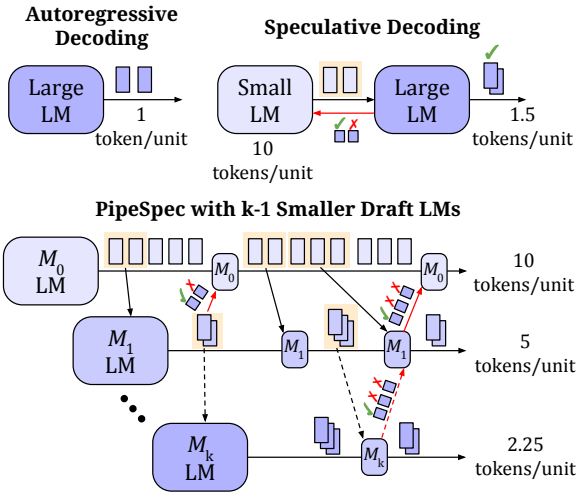

Figure 1: Comparison of different LLM decoding approaches. Top Left: Traditional autoregressive decoding (1 token/unit). Top Right: Speculative decoding using a small draft model (10 tokens/unit) for parallel verification by a large model (1.5 tokens/unit). Bottom: Our PipeSpec framework with $k-1$ draft models in a pipeline feeding into the large model (M_k), achieving 2.25 tokens/unit through pipelined parallelism. Checkmarks (✓) show accepted predictions while crosses (✗) indicate rejections triggering pipeline rollbacks.

1 Introduction

Large language models (LLMs) have transformed natural language processing through their remarkable ability to understand and generate human-like text. However, the fundamental requirement of autoregressive token generation, where each token must be generated sequentially based on all previous tokens, creates significant performance bottlenecks. This limitation is particularly pronounced in modern LLMs with 100B or more parameters (Dubey et al., 2024), making real-time applications challenging. Recent advances in speculative decoding have shown promise by leveraging smaller, faster models to draft candidate tokens for verification by larger models. However, current approaches still face fundamental efficiency limits due to their strict sequential dependencies between draft and verification stages.

As illustrated in Figure 1, traditional autoregressive decoding using a single large model is limited to 1 token per unit time due to strict sequential dependencies. Standard speculative decoding improves throughput to 1.5 tokens per unit time by employing a small draft model (10 tokens/unit) to generate candidates for batch verification by the large model. However, this approach still suffers from alternating idle periods where either the draft or verify model must wait for the other to complete.

Our key insight is that these limitations can be overcome through pipelining of multiple models. PipeSpec introduces a novel k-model architecture

Findings of the Association for Computational Linguistics: ACL 2025, pages 12909–12920

where each consecutive pair of models operates in an asynchronous producer-consumer relationship. In the three-model configuration shown in Figure 1 (bottom), an initial small model (M_0) rapidly generates draft tokens (10 tokens/unit), which are progressively refined by a medium-sized model (M_1, 5 tokens/unit) before final verification by the large model (M_k, 2.25 tokens/unit). This hierarchical structure provides two key advantages: (1) each stage operates asynchronously, enabling continuous parallel execution without idle periods, and (2) the intermediate models provide higher-quality draft tokens compared to single-draft approaches while still benefiting from their own draft-verify speedups.

PipeSpec operates through optimistic execution, where each model generates tokens assuming downstream acceptance. When a model rejects a prediction (marked as ✗), it triggers a rollback cascade – all subsequent predictions in earlier pipeline stages must be discarded and regenerated. This enables PipeSpec to maintain higher throughput than traditional Speculative Decoding. The main contributions of this work are:

- A novel hierarchical pipeline architecture for speculative decoding that breaks traditional stage dependencies, enabling continuous parallel execution across k models of increasing size and accuracy
- An analytical model that derives expected token generation rates and steady-state verification probabilities for pipelined models, with a proof of improved throughput over autoregressive decoding
- A complete multi-GPU implementation with efficient inter-device communication and rollback mechanisms, validated through extensive experiments showing consistent speedup over existing state-of-the-art speculative decoding approaches

2 Related Work

2.1 LLM Inference Acceleration

LLM inference consists of two distinct computational phases: prefill and decode. The prefill phase processes the initial input prompt, computing attention across all input tokens with quadratic memory scaling. The decode phase generates new tokens sequentially, requiring attention computation only against previous tokens' cached key-value pairs, making it more computationally bounded than memory bounded.

Recent research has targeted hardware-level optimizations for both phases. For prefill, FlashAttention (Dao et al., 2022; Dao, 2023) optimizes attention computation through tiling and recomputation strategies, particularly important for long sequences where naive implementations would exceed GPU memory bandwidth. Other approaches focus on GPU utilization (Hong et al., 2023; Vaidya et al., 2023; Patel et al., 2024) and efficient key-value cache management (Aminabadi et al., 2022; Sheng et al., 2023; Kwon et al., 2023). While these approaches optimize individual model execution, they are complementary to our proposed PipeSpec framework, which focuses on algorithmic speedups through pipelined speculative execution.

2.2 Speculative Decoding

While prefill optimizations like FlashAttention address the initial prompt processing, speculative decoding targets the decode phase bottleneck by leveraging parallel verification. First proposed by Stern et al. (Stern et al., 2018), the core idea is to use a smaller, faster draft model to generate multiple tokens sequentially that are then verified in parallel by the larger model, amortizing the cost of loading model weights and KV cache across multiple tokens (see Figure 1 top right).

Building on this foundation, researchers have developed various approaches to improve the efficiency of this draft-verify process. Tree-structured verification approaches (Miao et al., 2024; Li et al., 2024; Fu et al., 2024) expand beyond single-path prediction to explore multiple candidate sequences simultaneously, increasing the likelihood of successful verification of draft tokens. Other techniques like token distillation (Zhou et al., 2024), layer skipping (Zhang et al., 2023; Elhoushi et al., 2024), and retrieval-augmented drafting (He et al., 2024) aim to enhance draft model quality while maintaining low computational overhead. The MEDUSA framework (Cai et al., 2024) introduced specialized decoding heads to improve drafting efficiency without requiring a separate draft model; notably, all these algorithmic approaches are orthogonal to and could be combined with our systems-level pipeline optimization strategy.

More recently, several approaches have explored using multiple draft models to further accelerate inference. TRIFORCE (Sun et al., 2024) focuses specifically on extremely long-sequence generation (e.g., 100k context windows) by using the original

model with partial KV cache as an intermediate draft stage. Spector and Ré (Spector and Re) explored tree-structured batches across multiple draft models, though their approach remains tied to synchronous execution between stages. Our evaluation in Section 4 includes tiered speculative decoding configurations (using multiple draft models in sequence) which capture some of these benefits, but PipeSpec's key innovation is introducing true asynchronous pipelining where each model pair operates independently in a producer-consumer relationship. This fundamental architectural difference enables significantly higher throughput by maximizing hardware utilization across all available models, as demonstrated in our results.

2.3 Pipelined and Asynchronous Execution in Speculative Decoding

While speculative decoding has seen various advancements, PipeSpec's contribution centers on generalizing speculative decoding to an asynchronous, hierarchical pipeline designed for k independent, off-the-shelf models. This approach seeks to mitigate sequential dependencies and improve hardware utilization compared to synchronous or strictly two-stage speculative methods.

Several recent works have explored multi-model or parallel execution strategies. For instance, cascaded drafting approaches (e.g., Staged Speculative Decoding (Spector and Re), Cascade Speculative Drafting (Chen et al., 2024)) also utilize multiple models. These methods often focus on specific cascading structures or may retain some synchronous elements between stages. Similarly, techniques like Lookahead Decoding (Fu et al., 2024) aim to break sequential dependencies through different mechanisms, such as solving systems of equations for future tokens, rather than a pipelined multi-model execution.

The benefits of asynchronous execution in two-model (draft-verify) setups have been investigated (e.g., AMUSD (McDanel, 2024) and PipeInfer (Butler et al., 2024) primarily for multi-node distribution, PEARL (Liu et al., 2025) for adaptive draft length within a two-model parallel structure). PipeSpec extends the principle of asynchronous operation to a deeper hierarchy, where intermediate models in a k-stage chain concurrently verify outputs from predecessors and draft inputs for successors. This hierarchical and asynchronous coordination is a key aspect of our framework. SEED (Wang et al., 2024), for instance, also implements pipeline execution but focuses on breadth-wise parallelization for reasoning tree construction, differing from PipeSpec's depth-wise, single-input pipelining.

Other highly effective techniques, such as MEDUSA (Cai et al., 2024) or EAGLE-2 (Li et al., 2024), achieve significant speedups by training specialized prediction heads or employing dynamic draft trees. These methods often focus on enhancing the *quality and structure* of draft proposals, potentially requiring model-specific modifications or dedicated training phases. In contrast, PipeSpec is designed as a *systems-level execution paradigm* that can operate with existing, pre-trained models without necessitating additional training. The architectural design aims for continuous, overlapped computation across multiple models to reduce idle times (Section 4.5).

The concept of hierarchical speculation was noted as a potential extension in earlier work (e.g., by (Leviathan et al., 2023)). PipeSpec offers a concrete implementation of such an asynchronous multi-stage pipeline, supported by a theoretical analysis of its potential benefits (Section 3.3) and empirical results (Section 4.2). While complementary to methods that improve draft generation quality, PipeSpec's focus on an asynchronous, pipelined orchestration of independent models provides a distinct approach to LLM inference acceleration.

3 Hierarchical Pipelined Speculative Decoding

In this section, we first describe the operation of Hierarchical Pipelined Speculative Decoding (PipeSpec) as a k-stage pipeline (Section 3.1). We then present the core algorithm of PipeSpec (Section 3.2). Finally, we develop a theoretical framework to analyze PipeSpec's performance characteristics and compare it with existing approaches (Section 3.3).

3.1 Overview

Figure 2 compares token generation across different decoding approaches. The simplest approach, autoregressive decoding (top), uses a single large model (M_2) to generate tokens one at a time, achieving a throughput of 1 token per time unit. Traditional speculative decoding (second row) improves upon this by using a small draft model (M_0) that can generate tokens 4 times faster than M_2. However, despite this theoretical speedup, two key limitations prevent the system from achieving its

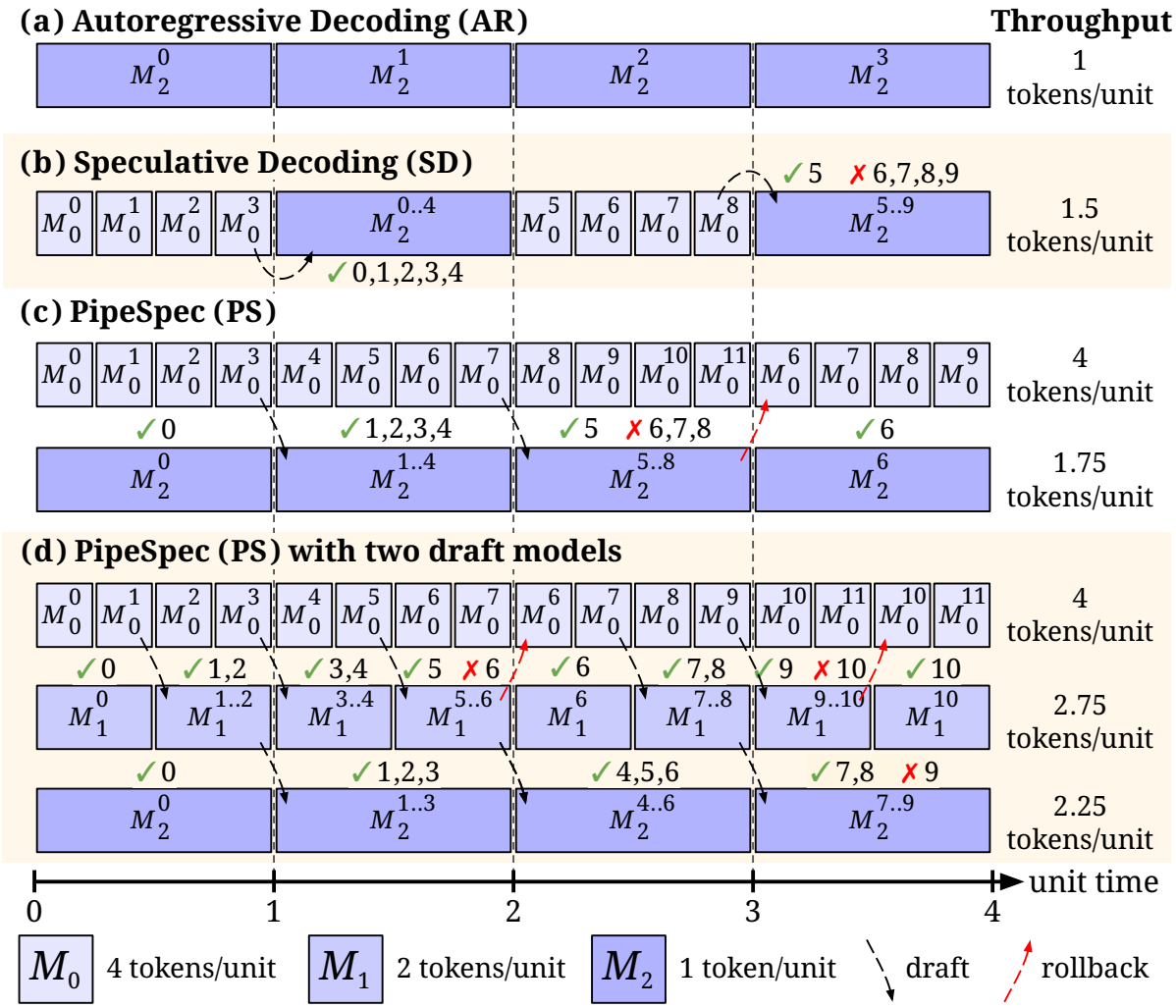

Figure 2: Comparison of different decoding approaches showing token generation over time. From top to bottom: (1) Traditional autoregressive decoding (AR) with sequential token generation using a single model M_2, (2) Standard speculative decoding (SD) using a draft model M_0 to generate candidate tokens verified in batches by M_2, (3) PipeSpec (PS) with 2 models showing continuous parallel execution between M_0 and M_2, and (4) PipeSpec with 3 models demonstrating hierarchical speculation across $\{M_0, M_1, M_2\}$.

full potential:

1. **Synchronous Execution:** The draft and verify stages operate in strict lockstep—M_0 must wait for M_2 to complete verification before generating the next batch of tokens. This creates alternating idle periods where M_0 is blocked waiting for verification results, and periods where M_2 is idle while new draft tokens are generated.
2. **Misprediction Penalty:** When M_2 rejects a prediction (marked with ✗ in the figure), all subsequent draft tokens in that batch become invalid and must be discarded. For example, in Figure 2(b), the rejection of token 6 invalidates the draft work done for tokens 7, 8, and 9, incurring a significant misprediction penalty.

These limitations combine to reduce the effective throughput to 1.5 tokens per unit, far below the theoretical maximum of the draft model.

PipeSpec introduces two key architectural innovations to address these inefficiencies. First, in its basic two-model configuration, we eliminate artificial synchronization requirements between draft and verification stages. This allows M_0 to optimistically generate additional draft tokens while M_2 verifies the prior batch of tokens in parallel. Assuming all tokens are accepted, the next verification stage can start immediately with a new batch of tokens, leading to improved throughput. However, when draft predictions are rejected, the system still needs to trigger a targeted rollback (shown by red

dashed lines) and resumes generation from the last valid token.

While this two-model configuration addresses the synchronization problem, misprediction penalties still impact performance significantly. To mitigate this, we introduce a three-model configuration with an intermediate model (M_1) that reduces misprediction penalties in two ways: (1) it quickly filters out low-quality predictions from M_0 before they reach the expensive M_2 verification stage, and (2) it provides M_2 with higher-quality draft tokens that are less likely to be rejected. This hierarchical refinement enables M_1 to serve as both a lightweight verification stage for M_0 and an improved draft model for M_2, achieving 2.25 tokens per unit (9 tokens verified in 4 time units) in Figure 2(d) while maintaining continuous parallel execution. This pipeline structure naturally extends to additional stages, with each intermediate model further reducing misprediction penalties through progressive refinement.

3.2 Algorithm

Algorithm 1 presents the core mechanism of Pipelined Speculative Decoding (PipeSpec). Each model i in our K-model pipeline maintains its own output buffer O_i, operating asynchronously while coordinating through a lightweight rejection mechanism. The first model ($i = 0$) continuously generates draft tokens, while verification models ($i > 0$) compare incoming draft tokens against their own token predictions. When a verification model rejects tokens (due to prediction mismatch), it signals earlier stages to rollback their buffers O_i to maintain consistency. The pipeline terminates when the final model O_K generates an end token, ensuring all tokens have been properly verified through the complete pipeline.

3.3 Theoretical Performance Analysis

Let $\mathcal{M} = \{M_0, M_1, \ldots, M_K\}$ represent a collection of LLMs ordered in increasing size, with t_i denoting the per-token generation time for model M_i. For any consecutive pair of models M_i and M_{i+1}, the token acceptance rate $\alpha_{i,i+1}$ is the probability that tokens generated by M_i are accepted by M_{i+1} during verification. In a hierarchical speculative decoding framework with K stages, the draft model $\mathcal{M}_{draft}$ can be any model from $\{M_0, \ldots, M_{K-1}\}$, while the target model $\mathcal{M}_{target}$ is M_K. The expected number of tokens $N(M_i)$ generated at M_i at each decoding step is then defined as:

$$E(N(M_i)) = (1 - \rho_i) \cdot 1 + \rho_i \cdot \frac{1 - \alpha_{i-1,i}^{\gamma_i+1}}{1 - \alpha_{i-1,i}} \quad (1)$$

where ρ_i represents the probability that M_i verifies the draft tokens generated by M_{i-1} in the window. γ_i represents the number of draft tokens from M_{i-1} that M_i attempts to verify in a single step (in PipeSpec, this is dynamically determined by the tokens available in M_{i-1}'s output buffer, unlike a fixed pre-set window in standard Speculative Decoding), and $\alpha_{i-1,i}$ is the probability that a token from M_{i-1} is successfully verified by M_i. If any draft token is rejected, the verification model generates one token in the next step, as illustrated in the PipeSpec workflow in Figure 2. Otherwise, $\frac{1-\alpha_{i-1,i}^{\gamma_i+1}}{1-\alpha_{i-1,i}}$ tokens will be produced, as derived from (Leviathan et al., 2023).

ρ_i should represent the probability of a steady state, because its calculation needs to take into account all previous token generation conditions of M_i up to the current step. To enter the verification process, one of the following two conditions must be met: if no verification was performed last time, the first draft token to be verified generated by M_{i-1} is consistent with the new token generated by M_i last time. Alternatively, if verification was performed last time, all draft tokens must pass, then the first draft token to be verified is consistent with the new token generated by M_i in the last step. Given $T = (t_0, t_1, \ldots, t_n)$, where t_j represents the j-th token generation step, the model M_i has performed calculations up to time step t_j. $\rho_i(t_j)$ represents the probability that M_i will do

Algorithm 1 Pipelined Speculative Decoding

Require: Input prompt, Models $[M_0 ... M_K]$
Ensure: Generated sequence O_K
Let O_i be token buffer for model i with length $|O_i|$
while not finished generating **do**
  for each model i running in parallel **do**
    if received rejection from stage $j > i$ **then**
      Rollback O_i to match O_j's last token
    if $i = 0$ **then** ▷ First generates drafts
      Generate next token, append to O_0
    else ▷ Others verify drafts
      Get draft tokens from O_{i-1}
      Generate token predictions
      Compare against predicted tokens
      Append matching tokens to O_i
      if any tokens do not match predictions **then**
        Signal rejection to earlier stages
  if end token in O_K **then**
    break
return O_K

verification at its j-th time step, which satisfies the following recursive condition

$$\rho_i(t_j) = \rho_i(t_{j-1}) \cdot \alpha_{i-1,i}^{\gamma_i+1} + (1 - \rho_i(t_{j-1})) \cdot \alpha_{i-1,i} \tag{2}$$

for $j > 1$, and we also have $\rho_i(t_0) = \alpha_{i-1,i}$.

According to the recursive equation 2, when $(n \to \infty)$, ρ_i reaches its stable state, which is given by,

$$\rho_i = \lim_{n\to\infty} \frac{1}{n+1} \sum_{j=0}^{n} \rho_i(t_j) = \frac{\alpha_{i-1,i}}{1 - \alpha_{i-1,i}^{\gamma_i+1} + \alpha_{i-1,i}} \tag{3}$$

Theorem 1 *For any $0 < \alpha < 1$ and $0 < \gamma$, the PipeSpec scheme generates a higher number of tokens per step.*

$$PipeSpec(P) = (1-\rho_k)\cdot 1 + \rho_k \cdot \frac{1 - \alpha_{k-1,k}^{\gamma_k+1}}{1 - \alpha_{k-1,k}} > 1 \tag{4}$$

It is obvious that $PipeSpec(P)$ is greater than 1 for any α and γ greater than 0, so the pipeline specification is definitely better than autoregressive decoding.

As for standard speculative decoding, we assume a two stage configuration $P_a = (M_d, M_t)$, where M_d represents draft model, M_t represents verification model. The acceptance rate for M_d and M_t is represented by $\alpha_{d,t}$, while the window size is given as γ_t. Additionally, $c_{d,t}$ denotes the speed ratio between the two models. Since each verification requires waiting for M_d to generate γ_t draft tokens, an additional $\frac{\gamma_t}{c_{d,t}}$ units of time are spent generating these draft tokens. Therefore, we can obtain the theoretical speedup of standard speculative decoding.

$$SD(P_a) = \frac{1 - \alpha_{d,t}^{\gamma_t+1}}{(1 - \alpha_{d,t})\left(\frac{\gamma_t}{c_{d,t}} + 1\right)} \tag{5}$$

If $\alpha_{d,t}$ is low, standard speculative decoding performs worse than autoregressive decoding due to the combined overhead of waiting for draft tokens and frequent verification failures. PipeSpec outperforms standard speculative decoding in this scenario since it eliminates waiting times through asynchronous execution. When $\alpha_{d,t}$ approaches its ideal case (higher acceptance rates), PipeSpec's theoretical performance improvement can be approximated as:

$$PipeSpec(P_a) \approx \frac{1 - \alpha_{d,t}^{\gamma_t+1}}{1 - \alpha_{d,t}} \tag{6}$$

Since PipeSpec does not need to spend time waiting for the draft models to generate draft tokens, it clearly has better performance than standard speculative decoding. The relationship between acceptance rate $\alpha_{d,t}$ and throughput is particularly evident in our HumanEval results, where the {1B, 8B, 70B} pipeline demonstrates how intermediate model refinement improves acceptance rates. This empirical improvement validates our theoretical prediction that pipeline depth can correlate positively with efficiency gains. For instance, on HumanEval using LLaMA3.1-70B (Table 2), a three-stage PipeSpec configuration {1B, 8B, 70B} achieves a 2.74× speedup, which is approximately 15% higher than the 2.38× speedup of a two-stage PipeSpec configuration {8B, 70B}. This highlights the benefit of an additional, smaller initial draft model (1B) refining tokens for the intermediate (8B) drafter, ultimately improving the quality of drafts presented to the final 70B verifier and increasing overall pipeline throughput. Each intermediate stage thus acts as both a verification filter and an improved draft model for subsequent stages.

4 Evaluation

Our evaluation examines four aspects: end-to-end performance across summarization and code generation tasks (4.2), token acceptance patterns and timing characteristics (4.3), the impact of lookahead window sizes on throughput (4.4), and GPU resource utilization (4.5).

4.1 Experimental Setup

All experiments were conducted on four NVIDIA A100-40GB GPUs interconnected via NVLink. GPU performance metrics were collected using nvidia-smi with 100ms sampling intervals.

We evaluated on the CNN/DM (Nallapati et al., 2016) and XSUM (Narayan et al., 2018) text summarization datasets, GSM8K (Cobbe et al., 2021) for mathematical reasoning, and the HumanEval (Chen et al., 2021) code generation benchmark. For models, we employed LLaMA-2 (Touvron et al., 2023) and LLaMA-3 (Dubey et al., 2024) variants, with each model allocated to dedicated GPU(s). The 70B variants used 4-bit quan-

tization and were split across 2 GPUs via pipeline parallelism. All experiments used greedy decoding (temperature=0.0) with maximum sequence length of 512 tokens, following prior work (Zhang et al., 2023; Elhoushi et al., 2024), to ensure a fair comparison.

4.2 Performance Analysis

Table 2 demonstrates the performance advantages of PipeSpec across multiple datasets and model configurations. The notation $\{M_0, M_1, ..., M_k\}$ in the Models column denotes a pipeline of models where M_0 is the smallest/fastest model and M_k is the verifier model. In traditional speculative decoding, these models operate sequentially – each model must wait for draft tokens from the previous model before beginning generation. In contrast, PipeSpec allows these models to operate asynchronously as discussed earlier in Section 3.1. Our evaluation reveals several significant trends:

First, PipeSpec consistently outperforms standard speculative decoding when using identical model configurations. For example, with a {68M, 7B} configuration on CNN/DM, PipeSpec achieves a 1.25× speedup compared to 1.11× for standard speculative decoding. This advantage becomes more pronounced with larger models - on HumanEval using LLaMA3.1-70B, PipeSpec with {8B, 70B} achieves 2.38× speedup versus 1.21× for speculative decoding.

Second, the results demonstrate clear benefits from longer pipeline configurations. On XSum using LLaMA2-13B, PipeSpec with three models {68M, 7B, 13B} achieves 2.03× speedup, significantly outperforming the two-model {68M, 13B} configuration at 1.74×. This is also shown for HumanEval using LLaMA3.1-70B, where extending the pipeline from {8B, 70B} to {1B, 8B, 70B} improves speedup from 2.38× to 2.74×. These results validate our theoretical analysis showing that pipeline efficiency increases with depth.

To better understand the contributions of PipeSpec's key architectural innovations, we conducted an ablation study on HumanEval using our LLaMA3.1-70B configuration, shown in Table 1. Disabling asynchronous pipeline execution (forcing synchronous stage dependencies) reduces speedup from 2.74× to 1.24×, highlighting the critical importance of breaking traditional stage dependencies. This substantial performance drop aligns with our theoretical analysis in Section 3.3, which predicted that eliminating synchronization overhead would be the primary driver of PipeSpec's advantages over traditional speculative decoding approaches.

Table 1: Impact of asynchronous pipeline execution and hierarchical model refinement on throughput using LLaMA3.1-70B on HumanEval. Speedup is relative to autoregressive baseline.

	Hierarchical Model Pipeline	
	Single Draft	Multi-Draft
Synchronous	1.21×	1.24×
Asynchronous	2.38×	2.74×

Similarly, using only a single draft model instead of our hierarchical pipeline drops performance to 2.38× under asynchronous execution, demonstrating the value of progressive token refinement through intermediate models. The baseline configuration with both synchronous execution and single draft model (effectively standard speculative decoding) achieves only 1.21× speedup, validating our architectural decision to pursue both asynchronous execution and hierarchical refinement in the full PipeSpec framework.

Finally, PipeSpec achieves competitive or superior performance compared to more complex algorithmic approaches like LayerSkip (Elhoushi et al., 2024) and Draft&Verify (Zhang et al., 2023), despite these methods employing sophisticated model-specific optimizations or additional pre-training. For instance, on CNN/DM using LLaMA2-13B, PipeSpec achieves 1.71× speedup compared to 1.81× for LayerSkip. Since these methods optimize different aspects of the inference process, they could potentially be combined with PipeSpec's asynchronous pipelining to achieve even greater speedups. (Note that speedup numbers for related works are taken from their original papers, though we use identical verifier model configurations and sizes for fair comparison.)

4.3 Token Generation Distribution and Timing Analysis

Figure 3 presents a comparative analysis of token acceptance patterns between speculative decoding (SD) and PipeSpec (PS) across different model configurations, aggregated across all samples in the HumanEval dataset. The top portion shows the frequency distribution of accepted tokens per step by the verify model, while the bottom portion shows the average time per token.

Table 2: Performance across decoding strategies. Speedup is relative to Autoregressive (AR) baseline. Time is in milliseconds/token. PipeSpec is our method.

	Method	Models	Time	Speedup
CNN/DM	AR Baseline	LLaMA2-7B	21.66	1.00×
	Speculative	68M,7B	19.49	1.11×
	LayerSkip	LLaMA2-7B	–	**1.86**×
	PipeSpec	68M,7B	17.35	1.25×
	AR Baseline	LLaMA2-13B	30.41	1.00×
	Speculative	68M,13B	25.23	1.21×
	Speculative	68M,7B,13B	23.80	1.28×
	Draft&Verify	LLaMA2-13B	–	1.56×
	LayerSkip	LLaMA2-13B	–	**1.81**×
	PipeSpec	68M,13B	22.59	1.35×
	PipeSpec	68M,7B,13B	17.74	1.71×
XSum	AR Baseline	LLaMA2-7B	21.85	1.00×
	Speculative	68M,7B	15.13	1.44×
	LayerSkip	LLaMA2-7B	–	1.54×
	PipeSpec	68M,7B	**12.92**	**1.69**×
	AR Baseline	LLaMA2-13B	29.69	1.00×
	Speculative	68M,13B	18.43	1.61×
	Speculative	68M,7B,13B	23.20	1.28×
	Draft&Verify	LLaMA2-13B	–	1.43×
	LayerSkip	LLaMA2-13B	–	1.48×
	PipeSpec	68M,13B	17.07	1.74×
	PipeSpec	68M,7B,13B	**14.64**	**2.03**×
GSM8K	AR Baseline	LLaMA2-7B	20.91	1.00×
	Speculative	68M,7B	7.32	2.86×
	PipeSpec	68M,7B	**6.24**	**3.35**×
	AR Baseline	LLaMA2-13B	27.75	1.00×
	Speculative	68M,13B	9.59	2.89×
	Speculative	68M,7B,13B	18.06	1.54×
	PipeSpec	68M,13B	**8.60**	**3.23**×
	PipeSpec	68M,7B,13B	9.01	3.08×
HumanEval	AR Baseline	LLaMA2-13B	28.12	1.00×
	Speculative	68M,13B	17.57	1.60×
	Speculative	68M,7B,13B	23.46	1.20×
	Draft&Verify	CLLaMA2-13B	–	1.46×
	LayerSkip	LLaMA2-13B	–	1.66×
	PipeSpec	68M,13B	17.24	1.63×
	PipeSpec	68M,7B,13B	**14.85**	**1.89**×
	AR Baseline	LLaMA3.1-70B	110.31	1.00×
	Speculative	8B,70B	91.21	1.21×
	Speculative	1B,8B,70B	88.78	1.24×
	PipeSpec	8B,70B	46.34	2.38×
	PipeSpec	1B,8B,70B	**40.31**	**2.74**×

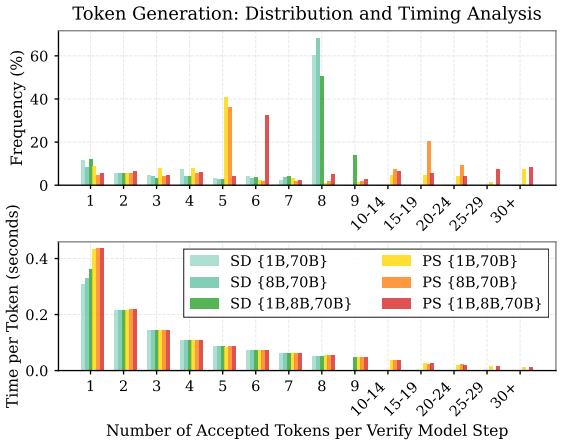

Figure 3: Analysis of token acceptance patterns and timing across decoding strategies on HumanEval. Top: Distribution of accepted tokens per verify step, showing SD's fixed window behavior versus PipeSpec's more flexible patterns. Bottom: Average time per token as a function of batch size, demonstrating PipeSpec's minimal synchronization overhead.

SD exhibits a pronounced spike at 8 tokens per verification step across all configurations, resulting from its fixed lookahead window size. This creates a rigid operational pattern where SD must strictly alternate between drafting and verifying batches of 8 tokens, balancing between batch processing efficiency and computational waste.

PipeSpec exhibits a notable long-tail distribution in token acceptance patterns, with successful verifications extending well beyond 20 tokens in both two-model PS {1B, 70B} and three-model PS {1B, 8B, 70B} configurations. The asynchronous design enables natural acceptance patterns to manifest, with a distinctive spike at 6 tokens in the three-model setup emerging from pipeline stage optimizations. This flexibility, combined with the intermediate model's filtering effect, facilitates larger batch sizes by efficiently discarding lower-quality predictions before they reach the computationally intensive verification stage at 70B. This long-tail distribution indicates that PipeSpec can effectively capitalize on 'easy' prediction sequences where models agree, allowing significantly larger sequences to be processed when token predictions align well, while still maintaining fast recovery through the pipeline when predictions diverge.

4.4 Token Lookahead Analysis

As shown in Figure 4, the lookahead window size (the number of tokens generated by draft models

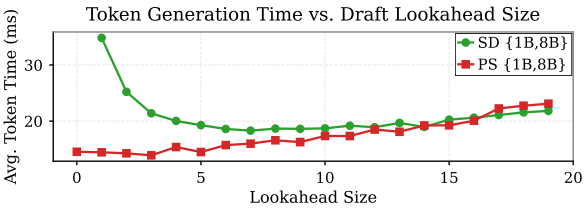

Figure 4: Impact of lookahead window size on token generation time. SD shows poor performance at small windows due to synchronization overhead and at large windows due to wasted speculation. PS maintains lower latency at small windows but degrades at larger sizes as verification must wait for draft tokens.

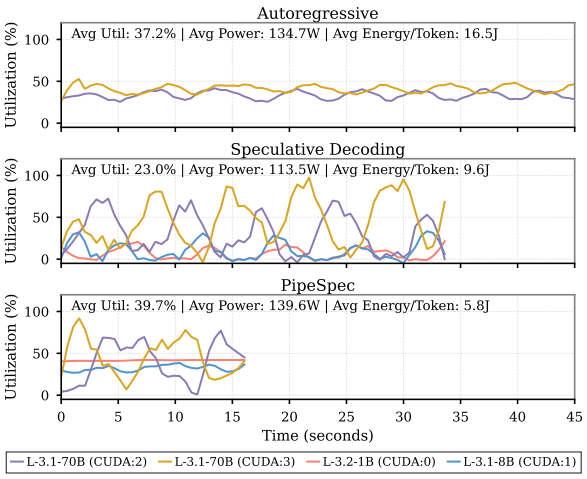

Figure 5: GPU utilization over time showing autoregressive (70B model split across 2 GPUs), speculative decoding ({1B,8B,70B}), and PipeSpec ({1B,8B,70B}). PipeSpec achieves higher average utilization (39.7%) by eliminating idle periods between draft and verification.

before verification) significantly shapes the performance characteristics of both approaches. For SD, small windows (1-5 tokens) lead to high latency as the verify model lacks sufficient tokens to batch process effectively, while moderate windows (5-10 tokens) improve performance through better batching before degrading beyond 10 tokens due to increased speculation waste. In contrast, PS maintains lower latency at small window sizes through continuous pipeline processing, though it also experiences degradation with larger windows as verification must wait for more draft tokens to accumulate. These results reveal different optimal operating points. SD performs best with moderate lookahead windows (8-10 tokens), while PS achieves optimal performance with minimal lookahead. For SD experiments, we used a fixed lookahead window of 8 tokens. PipeSpec operates by processing available draft tokens immediately. As shown in Figure 4, PipeSpec achieves its best performance with a minimal, dynamically determined lookahead (i.e., verifying tokens as soon as they are drafted by the preceding stage), while forcing larger, fixed lookahead windows degrades its performance.

4.5 Resource Utilization

Figure 5 shows GPU utilization patterns across decoding approaches for a HumanEval sample using LLaMA3.1-70B. While autoregressive decoding achieves 37.2% utilization, traditional speculative decoding exhibits pronounced idle periods where draft models drop to near-zero utilization while awaiting verification, resulting in 23.0% average utilization. PipeSpec maintains consistently higher GPU activity (39.7%) through pipelining, eliminating these idle periods. This improved hardware utilization translates to better energy efficiency, with PipeSpec achieving 5.8J/token compared to 16.5J/token for autoregressive decoding.

5 Conclusion

We introduced PipeSpec, a novel framework that fundamentally rethinks speculative decoding by breaking traditional sequential dependencies in LLM inference through asynchronous, hierarchical pipelined execution. Our analytical model not only proves guaranteed throughput improvements over autoregressive decoding for any non-zero acceptance rate but also provides insights into the steady-state behavior of such pipelined systems. Empirically, PipeSpec consistently delivers substantial speedups, often achieving two to three-fold improvements over autoregressive baselines and notable gains over standard speculative decoding across various models and tasks. This performance is achieved by effectively utilizing multi-GPU setups and minimizing idle hardware time.

A key finding is that pipeline efficiency often increases with depth; for instance, three-model configurations generally outperformed two-model setups by enabling progressive token refinement and better draft quality for the final, largest model. This suggests a promising architectural paradigm for future inference systems, particularly as models continue to grow in scale and complexity. By facilitating continuous parallel execution and adaptable draft quality through its hierarchical structure, PipeSpec offers a robust and scalable approach to accelerating LLM inference.

Limitations

A key limitation of PipeSpec lies in its static pipeline configuration strategy. The current approach uses fixed model selections and predetermined pipeline depths, which may not be optimal across different tasks or input characteristics. Some generation tasks might benefit from deeper pipelines with more intermediate verification stages, while others might achieve better performance with shallower configurations. The system lacks mechanisms to dynamically adjust its architecture based on task complexity, resource availability, and observed prediction patterns. This rigidity means PipeSpec cannot adapt to changing computational demands or leverage emerging patterns in token generation that might suggest more efficient pipeline arrangements.

From an implementation perspective, the system's performance is heavily dependent on the quality of draft model predictions. While our hierarchical approach helps mitigate poor predictions through progressive refinement, frequent mispredictions can still trigger expensive rollback cascades across multiple pipeline stages. The current design assumes all models can fit within available GPU memory, with larger models split across devices as needed. This may not scale effectively to scenarios with more severe memory constraints or when using very deep pipelines with many intermediate models. Additionally, while PipeSpec reduces overall inference latency and can achieve lower energy *per generated token* (as shown in Section 4.5) compared to autoregressive decoding due to efficient batched verification, its concurrent operation of multiple models inherently requires more hardware resources. This leads to higher *instantaneous power draw*. The overall energy efficiency depends on maintaining high token acceptance rates; frequent rollbacks could diminish the per-token energy benefits compared to a highly optimized single-model baseline. The continuous parallel execution across multiple GPUs leads to higher sustained power draw, raising important questions about the trade-offs between speed and efficiency as language models continue to grow in size and complexity. Specifically, for our largest model configurations (e.g., LLaMA3.1-70B), the verifier was deployed using pipeline parallelism. An autoregressive baseline for this 70B model, if fully optimized with tensor parallelism, would likely achieve higher throughput. Consequently, while PipeSpec's verifier could also leverage tensor parallelism, the precise net speedup attributable to PipeSpec's hierarchical asynchronous architecture in these specific large-model scenarios would require a re-evaluation against such an optimally parallelized baseline, which we defer to future work.

An important consideration for future work involves the interaction between decoding strategies and model behavior in evaluation settings. Our experiments, following established benchmarking protocols from prior speculative decoding research, employed greedy decoding (temperature=0.0) with extended sequence lengths (512 tokens) to ensure fair comparison across methods. However, we observed that greedy decoding can sometimes lead to repetitive generation patterns, particularly in summarization tasks where models may continue regenerating portions of the original article beyond the intended summary. While this behavior is consistent across all evaluated methods and actually tends to favor speculative approaches (as draft models can more easily predict such repetitive patterns), it raises important questions about evaluation methodology in acceleration research. The enhanced speedups observed in longer sequence settings may partially reflect this predictable repetition rather than solely representing improvements in meaningful content generation. Future work should consider evaluating speculative decoding methods across diverse decoding strategies (including sampling-based approaches) and with more nuanced metrics that distinguish between productive and repetitive generation, to provide a more comprehensive understanding of their real-world applicability.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Branden Butler, Sixing Yu, Arya Mazaheri, and Ali Jannesari. 2024. Pipeinfer: Accelerating llm inference using asynchronous pipelined speculation. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–19. IEEE.

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. *arXiv preprint arXiv:2401.10774*.

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

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chang, and Jie Huang. 2024. Cascade speculative drafting for even faster llm inference. *Advances in Neural Information Processing Systems*, 37:86226–86242.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*, 9.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 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, et al. 2024. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

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 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.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. 2023. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*.

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*, pages 611–626.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. *Preprint*, arXiv:2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. 2025. Pearl: Parallel speculative decoding with adaptive draft length. In *The Thirteenth International Conference on Learning Representations*.

Bradley McDanel. 2024. Amusd: Asynchronous multi-device speculative decoding for llm acceleration. *arXiv preprint arXiv:2410.17375*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 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*, pages 932–949.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Benjamin Frederick Spector and Christopher Re. Accelerating llm inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models@ ICML2023*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, pages 10107–10116.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Neal Vaidya, Fred Oh, and Nick Comly. 2023. Optimizing inference on large language models with nvidia tensorrt-llm, now publicly available. https://github.com/NVIDIA/TensorRT-LLM. [Online].

Zhenglin Wang, Jialong Wu, Yilong Lai, Congzhi Zhang, and Deyu Zhou. 2024. Seed: Accelerating reasoning tree construction via scheduled speculative decoding. *arXiv preprint arXiv:2406.18200*.

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*.

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*.