PRISM: PARAMETRICALLY REFACTOR INFERENCE FOR SPECULATIVE DECODING DRAFT MODELS

Xuliang Wang [*1 2] **Yuetao Chen** [*3] **Maochan Zhen** [2] **Fang Liu** [2] **Xinzhou Zheng** [4] **Xingwu Liu** [5 2] **Hong Xu** [3] **Ming Li** [1 2]

ABSTRACT

Large Language Models (LLMs), constrained by their auto-regressive nature, suffer from slow decoding. Speculative decoding methods have emerged as a promising solution to accelerate LLM decoding, attracting attention from both systems and AI research communities. Recently, the pursuit of better draft quality has driven a trend toward parametrically larger draft models, which inevitably introduces substantial computational overhead. While existing work attempts to balance the trade-off between prediction accuracy and compute latency, we address this fundamental dilemma through architectural innovation.

We propose PRISM, which disaggregates the computation of each predictive step across different parameter sets, refactoring the computational pathways of draft models to successfully decouple model capacity from inference cost. Through extensive experiments, we demonstrate that PRISM outperforms all existing draft architectures, achieving exceptional acceptance lengths while maintaining minimal draft latency for superior end-to-end speedup. We also re-examine scaling laws with PRISM, revealing that PRISM scales more effectively with expanding data volumes than other draft architectures. Through rigorous and fair comparison, we show that PRISM boosts the decoding throughput of an already highly optimized inference engine by more than 2.6×.

1 INTRODUCTION

Large Language Models (LLMs) have become a foundational technology. Models such as OpenAI's ChatGPT (OpenAI, 2022), Meta's LLaMA (Touvron et al., 2023), and Google's Gemini (Google, 2023) have demonstrated remarkable capabilities across a diverse range of tasks, proving potentials for novel applications that advance human-computer interaction (Masson et al., 2024), optimize industrial workflows (Xia et al., 2023), and create new paradigms for scientific endeavors (Jumper et al., 2021). However, realizing these potentials in practical applications is often constrained by the critical problem of generation efficiency.

The efficiency problem is inherent to the auto-regressive nature of LLM inference. Auto-regressive generations usually adopt a sequential scheme, performing a full forward pass through its parameters to produce each token, which results in substantial computational cost and low throughput.

To mitigate this problem, the research community has proposed a range of acceleration techniques. While optimizations such as efficient KV cache management (Kwon et al., 2023; Qin et al., 2025), continuous batching (Yu et al., 2022), parallelism optimization (Li et al., 2023; Butler et al., 2024), and prefill-decode disaggregation schemes (Zhong et al., 2024b;a; Patel et al., 2025) have yielded significant speedups, they do not alter the fundamental token-by-token generation paradigm. As a complementary approach, speculative decoding (Leviathan et al., 2023) effectively reduces the number of LLM forward passes.

Speculative decoding, also known as Draft-and-Verify, is inspired by speculative execution, a well-established optimization technique in modern CPU design. It primarily consists of two stages: in the first stage, a smaller, faster draft model proposes a sequence of candidate tokens; in the subsequent stage, the drafted sequence is verified by the larger, slower target model. Intuitively, when multiple drafted tokens are accepted, speculative decoding enables the target model to generate multiple tokens in a single forward pass, whereas naive auto-regressive decoding generates only one token per pass. Consequently, the success

[*]Equal contribution [1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada [2]Central China Institute of Artificial Intelligence, Zhengzhou, Henan, China [3]The Chinese University of Hong Kong, Hong Kong SAR, China [4]University of Science and Technology of China, Hefei, Anhui, China [5]Dalian University of Technology, Dalian, Liaoning, China. Correspondence to: Ming Li <mli@uwaterloo.ca>.

Preprint.

Table 1. A comparison of draft model relative sizes used in related works, measured in drafter parameter counts divided by target model sizes. Note that the statistics include all parameters activated in the forward pass and LLMs mentioned is the chat/Instruct Version. This table shows a growing trend in adopting relatively larger drafter models.

Paper	Base LLM	Drafter	Relative Size
Leviathan et al.	T5-XXL-11B	T5-SMALL (77M)	0.68%
Miao et al.	LLaMA-2-7B	LLaMA-68M	0.95%
	LLaMA-2-65B	LLaMA-68M	0.10%
Li et al.	LLaMA2-7B	EAGLE-2 (0.48B)	6.86%
	LLaMA2-70B	EAGLE-2 (1.23B)	1.76%
Li et al.	LLaMA3.1-8B	EAGLE-3 (1.03B)	12.88%
	LLaMA3.3-70B	EAGLE-3 (2.19B)	3.13%
Tang et al.*	LLaMA-3.3-8B	Scaled EAGLE-2 (1.59B)	19.88%
	Llama-3.3-70B	Scaled EAGLE-2 (4.35B)	6.21%
Yan et al.*	LLaMA-2-7B	Scaled EAGLE-2 (1.26B)	18.00%
	LLaMA-2-13B	Scaled EAGLE-2 (1.78B)	13.69%

* We calculate drafter sizes for these rows as the drafter models are not publicly available.

of speculative decoding strongly relies on draft quality, i.e. , how many tokens proposed by the draft model are expected to be accepted by the target model in one draft.

To improve draft quality, many drafter models, following EAGLE (Li et al., 2024a), utilize the target model's intermediate hidden states from the previous verification step as additional input. This enables more accurate drafts than drafter models that rely on tokens alone. Many of these draft models employ only one transformer block to minimize the computational overhead of drafting.

However, the single-transformer configuration limits draft model capacity and ultimately caps acceptance rates. Recognizing this limitation, as shown in Table 1, emerging works have begun to explore the viability of larger draft models (Li et al., 2025a; Yan et al., 2025; Tang et al., 2025). While maintaining a similar architecture, these works stacks more transformer layers. Although larger drafters do incur more computational overhead, the argument is that this cost is outweighed by the gains from improved acceptance rates.

In this paper, we investigate the fundamental trade-off between draft quality and computational overhead in speculative decoding from a different perspective. Our key insight is that while larger models are necessary for enhancing drafter predictive performance, it is possible to keep draft overhead low simultaneously. We demonstrate this through PRISM, which recognizes the inherent differences between draft steps and leverages conditional computing. Specifically, we refactor the inference computational path by distributing computation across different draft steps to distinct parameter sets, analogous to how a prism disperses white light into its spectrum. Consequently, the total parameter count of the drafter expands while the number of activated parameters per draft step remains constant. Fewer activated parameters accelerate both inference and training. Critically, the model's representational capacity still expands: through extensive experiments, we show that PRISM's architecture not only preserves favorable scaling properties but actually scales more effectively than naive parameter scaling approaches such as vertically stacking transformer layers.

In summary, this paper addresses the core trade-off in speculative decoding between drafter predictive quality and efficiency by introducing PRISM, a novel drafter architecture. Our contributions include:

- **A Novel Architecture that Decouples Draft Model Capacity from Draft Computational Cost.** We introduce PRISM, the first architecture to apply a conditional computing paradigm specifically across autoregressive generation steps to expand model capacity. Unlike conventional draft models where capacity and computational overhead are entangled, PRISM successfully decouples them by parametrically disaggregating the computation of different draft steps. This novel architecture propose new perspective for drafter effectiveness-efficiency trade-off, showing new paths for drafter design.
- **Re-examining the Scaling Law for Draft Models.** Our work provides the first empirical evidence that a draft model's predictive power can scale effectively without increasing the activated parameter count. This finding establishes a new, more efficient scaling paradigm for draft models.
- **Detailed Evalution with Inference Engine.** We bridge the gap between speculative decoding optimizations from the AI and the system academia community by implementing and validating PRISM within SGLang, a state-of-the-art inference engine. In contrast to many prior drafters evaluated solely in PyTorch-based settings, our experiments provide robust evidences of PRISM's efficacy with a system-level highly optimized inference engine.

2 BACKGROUND

This section provides a brief overview of LLM inference, speculative decoding and state-of-the-art drafter architectures.

LLM Inference. LLM inference is fundamentally autoregressive, probabilistically generating a sequence of tokens $T = (t_1, t_2, ..., t_n)$ where each token t_i is sampled from a conditional probability distribution parameterized by the model weights θ and the preceding tokens: $p(t_i|t_1, ..., t_{i-1}; \theta)$. This sequential dependency makes an iterative generation process necessary.

By computaional characteristics, LLM inference are bifurcated into two distinct phases, prefill and decoding.

The prefill phase refers to the initial phase involves the processing of the input prompt. The model performs a highly parallel forward pass to compute the attention mechanism's key-value (KV) pairs for all prompt tokens simultaneously. This phase is computation-bound, contributed by large matrix-matrix multiplications across the entire input sequence.

Following the prefill, the model enters the iterative decoding phase. Different from the prefill phase, the decoding phase is memory-bandwidth-bound as it necessitates loading the entire set of model parameters from off-chip high-bandwidth memory (HBM) to the on-chip processing units for each step. This memory-bandwidth bottleneck is further exacerbated in scenarios involving long contexts where with each generated token, the model must also read the attention KV caches for all preceding tokens.

Speculative Decoding. The core principle of Speculative Decoding involves using a parametrically smaller, faster draft model M_{draft} to generate a sequence of candidate tokens based on the current context, which are then parallelly validated in a single forward pass by the parametrically larger, slower target model M_{target}.

Formally, let $p_{target}(t|c)$ be the probability distribution over the next token t given a context c, as defined by M_{target}. Similarly, let $p_{draft}(t|c)$ be the distribution from M_{draft}. During drafting, At a given step with context c, M_{draft} is invoked to generate a candidate sequence of γ tokens, $\tilde{T} = (\tilde{t}_1, \tilde{t}_2, ..., \tilde{t}_\gamma)$. Then for verification, M_{target} performs a single forward pass on the concatenated input $(c, \tilde{T})$ to efficiently compute the true probability distributions for each position: $p_{target}(\cdot|c)$, $p_{target}(\cdot|c, \tilde{t}_1)$, ..., $p_{target}(\cdot|c, \tilde{t}_1, ..., \tilde{t}_{\gamma-1})$. The draft tokens are validated sequentially using a modified rejection sampling scheme: for each draft token $\tilde{t}_i$, where the initial value of i is 1, it is accepted if a randomly drawn number $r \in \text{Uniform}(0, 1)$ which satisfies

$$r < \min(1, \frac{p_{target}(\tilde{t}_i|c, \tilde{t}_{<i})}{p_{draft}(\tilde{t}_i|c, \tilde{t}_{<i})})$$

The process iterates over i, until all tokens are accepted or a token $\tilde{t}_k$ is rejected. All accepted tokens $\hat{T}$ are kept. Additionally, an extra token could be sampled from a corrected distribution $p'_{target} \propto \max(0, p_{target}(\cdot|c, \hat{T}) - p_{draft}(\cdot|c, \hat{T}))$. Following this, the model's context is updated with the newly generated tokens, the KV cache entries corresponding to any rejected candidates are discarded, and a new drafting cycle commences.

A fundamental and critical property of speculative decoding is that it is lossless. This has been formally proven and guaranteed that the final output sequence produced by the speculative process is drawn from the exact same probability distribution as if it were generated token-by-token by M_{target} alone (Miao et al., 2024).

From the system view, the efficacy of speculative decoding lies in its ability to amortize the high cost of the memory-bandwidth-bound decoding steps. The conventional one-token-per-pass auto-regressive scheme is heavily memory-bandwith-bound, leading to poor GPU utilization as the computing units stall waiting for data. Speculative decoding transforms the workload by using a single forward pass to verify multiple tokens in parallel. When the draft model's predictions align well with the target model's, leading to a high acceptance rate, the effective cost per token is drastically reduced. This mechanism fundamentally alters the operational characteristics of the decoding phase, increasing its compute-to-memory ratio.

SoTA Draft Models. Initial approaches to speculative decoding primarily utilized lightweight versions of the target LLM as draft models. A significant difference was introduced by EAGLE (Li et al., 2024a), who first leverages not only the preceding context tokens but also the hidden states generated by the target model during the most recent verification step. By conditioning on this more informative input, EAGLE's draft models can more accurately predict the target model's distribution, leading to a substantially higher acceptance rate for candidate tokens. The efficacy of exploiting informative intermediate representations from the target model has led to its adoption in subsequent works (Li et al., 2024b; 2025a; Huang et al., 2025; Zhang et al., 2025a).

Most SoTA draft models also utilize a tree-based Draft-and-Verify approach instead of the naive sequence-based approach. Initially proposed by Specinfer (Miao et al., 2024), the key observation is that even when the most likely drafted token is rejected, an alternative, less probable candidate from the same generation probability distribution might still be correct. By allowing multiple tokens to be validated for the same generation step, the draft tokens form different candidate paths and can be organized into a tree structure. During the verification phase, a special tree-shaped attention mask is applied to enforce the causal relationships among the draft tokens. Tree-based draft and verify significantly enhances acceptance rate.

Scaling of drafter models. The expectation of draft models has previously been governed by the principle of minimizing computational overhead, resulting in parametrically small drafters. This paradigm, however, inherently limits the model's capacity and its ability to benefit from large-scale training data. Progressively recognizing the limitation, recent research has shifted towards balancing the trade-off between draft model complexity and its computation latency. The goal is to identify a *sweet spot* where a more powerful, albeit more costly, draft model that yields a high enough acceptance rate to deliver high end-to-end acceleration.

EAGLE-3 (Li et al., 2025a) pioneered the study of intentionally scaling draft models, demonstrating that increasing trainable parameters and making other architectural refinements enabled considerable performance gains when trained with more data; Scylla (Yan et al., 2025) scaled the EAGLE-2 architecture with up to five trainable transformer layers, pretraining and tuning it on a considerably larger dataset, resulting in state-of-the-art acceptance lengths. In parallel, Meta (Tang et al., 2025) has explored different axes of scaling, investigating methods for parameter expansion such as increasing model depth and substituting standard Feed-Forward Networks (FFN) with Mixture-of-Experts (MoE) layers. These efforts collectively signal a transfer of expectations for draft models, moving from the minimal overhead to balanced performance optimized for maximum system throughput.

3 METHODOLOGY

3.1 Motivation

As the practicability of scaling draft models to achieve high acceptance rates emerges, it is compelling to train draft models with considerably more parameters. To expand model capacity without introducing extra overhead, we drew inspiration from the MoE models, which leverage specialization by routing computations to different sub-networks. We observe a similar opportunity for specialization exists within speculative decoding: the predictive task in speculative decoding is non-uniform in its difficulty. There is a clear empirical trend where drafting becomes progressively harder at later steps in the sequence. This is consistently demonstrated by the sharp decline in acceptance rates for later tokens (see Figure 1).

This insight directly motivates our proposed architecture, PRISM. Rather than relying on a single, one-size-fits-all model, PRISM employs a principle of specialization by applying different set of parameters to different draft steps. This design brings two benefits intuitively:

- **Adaptive Representation Complexity.** PRISM effectively creates a hierarchical model where later parameters process the output of previous ones, forming a cascaded computational structure. This design enables adaptive allocation of computational resources across draft steps. As shown in Figure 2, compared with previous architectures where all draft steps utilize the same set of parameters across draft steps, PRISM progressively increases the effective depth with draft steps. This architectural choice naturally allocates more cumulative computation to harder prediction tasks at the latter part of the sequence, where uncertainty typically increases and accurate token prediction becomes more challenging.

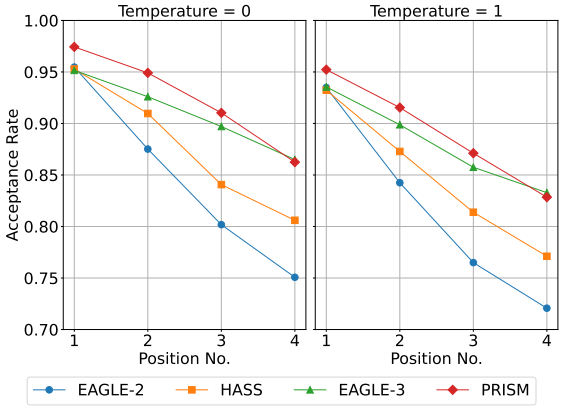

Figure 1. Emperical results showing the average acceptance rate step-wise on LLaMA-3-8B.

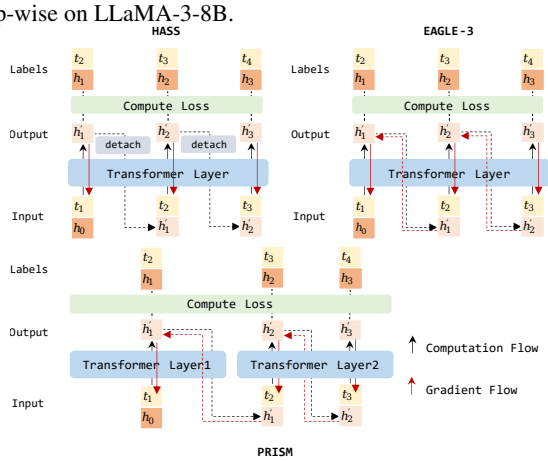

Figure 2. An illustration of computation and gradient flow of HASS, EAGLE-3 and PRISM.

- **Decoupling Model Capacity from Inference Cost.** Critically, this architecture decouples the model's total learning capacity from its per-step inference cost. We can substantially increase the total number of trainable parameters, allowing the model to learn more complex functions, while the computational cost of generating any single draft token remains constant and minimal.

Following the principle of conditional computation central to MoE models, PRISM is a *sparsely-activated* architecture that decouples total model capacity from per-step computational cost. To formalize this, let t denote the parameter count of a single processing module and n the number of modules in PRISM. Setting aside shared components that are common to all steps, the total parameter count is $P_{\text{total}} \approx n \cdot t$. Because exactly one module is selected per generation step, the model is *sparsely activated*: only $P_{\text{active}} \approx t$ parameters and the corresponding FLOPs are exercised per step. PRISM thus provides $n\times$ the parameter capacity of a single module while keeping per-step computation constant, analogous to how an n-expert MoE layer offers $n\times$ the capacity of

Table 2. Notation for PRISM components and variables.

Notation	Description
S_{acc}	Sequence of n accepted tokens from previous verification: $(x_1, ..., x_n)$
H_{base}	Target model hidden states for S_{acc}: $(h_0, h_1, ..., h_{n-1})$
E	Token embedding layer (frozen)
$L_1, ..., L_M$	M distinct Transformer layers
$F_i(\cdot, \cdot)$	Fusion function for L_i combining token embedding and hidden state
f_{map}	Surjection mapping K draft steps to M modules: $1, ..., K \to 1, ..., M$
LM_{head}	Vocabulary projection layer (frozen)

a single expert at a per-token cost of roughly one expert. By contrast, a conventionally stacked dense model with the same total parameter budget $n \cdot t$ activates all parameters for every token, so its per-step cost grows linearly with capacity and the capacity-to-cost ratio collapses to $1\times$.

3.2 PRISM Architecture

PRISM, or Parametrically Refactor Inference for Speculative draft Models, follows the SoTA Drafters' convention to be a transformer-based drafter. As shown in Figure 3, PRISM consists of a token embedding layer, a final vocabulary projection head and a sequence of processing modules. Each processing module consists of a fully connected fusion layer and a transformer layer.

The relationship between draft steps and processing modules in PRISM is defined by a surjection. We establish the surjection by assigning each draft step to a specific processing module for its computation. This mapping allows for a many-to-one relationship: while every step has a uniquely assigned module, a single processing module can serve multiple steps.

Draft with PRISM. The draft process is featured by a unique proactive parameter switching procedure between inference forward passes, as shown in Figure 3.

Formally, define the model's components and variables as in Table 2.

The draft starts with a prefill of the draft model. For each accepted token $x_i \in S_{acc}$, a fused representation is created by combining its embedding with the corresponding hidden state h_{i-1} from the target model and feeding it to the fusion function

$$H_{\text{fused}} = [F_{f_{map}(1)}(E(x_i), h_{i-1})]_{i=1}^{n}$$

This sequence of fused representations is then processed in a single forward pass through the designated transformer layer

$$H'_{out}, KV_{(1)} = L_{f_{map}(1)}(H_{fused}, KV_{(0)})$$

Where $KV_{(0)}$ is the initial state of the KV cache.The output consists of a hidden state sequence H'_{out} and the updated KV cache $KV_{(1)}$. Note that the KV caches will be shared across processing modules. The final hidden state from H'_{out}, i.e. h'_n, becomes the initial predictive state for the drafter auto-regressive decoding phase.

$$\hat{h}_0 = h'_n$$

The initial input token for the decoding phase $\hat{x}_1$ is also the first draft token. It should be predicted directly from $\hat{h}_0$ using the classification head

$$\hat{x}_1 = \arg\max(LM_{head}(\hat{h}_0))$$

The remaining draft steps are all decoding steps repeating similar procedures: for $k = 2, 3, ..., K$,

$$H^{(k)}_{in}, KV_{(k)} = F_{f_{map}(k)}(E(\hat{x}_{k-1}, \hat{h}_{k-2}), KV_{(k-1)})$$

$$\hat{h}_{k-1} = L_{f_{map}(k)}(H^{(k)}_{in})$$

$$\hat{x}_k = \arg\max(LM_{head}(\hat{h}_{k-1}))$$

The whole draft process ends with output draft tokens $(\hat{h}_1, \hat{h}_2, ..., \hat{h}_k)$, which will then sent to be verified.

Notably, the formulation above assumes a linear generation path. This is an simplification for the sake of presentation clarity. Actually, PRISM mechanism is totally compatible with tree-structured draft and stochastic sampling methods that many other drafters equip.

Train PRISM. To mitigate the problem of exposure bias, many drafters employ context alignment techniques during training, which significantly improves acceptance length. Prominent examples include the harmonized context alignment in HASS (Zhang et al., 2025a), the train-time test in EAGLE-3 (Li et al., 2025a), and token alignment in Griffin (Hu et al., 2025). Context alignment aligns input hidden states, KV caches and even input tokens from the training and the inference. With PRISM, we followed the HASS style of aligning the hidden states and KV caches for 3 steps.

Context alignment simulates the auto-regressive inference process in training by requiring a separate forward pass for each draft step. This significantly increases training cost: for instance, training EAGLE-3 to draft n tokens requires n distinct forward passes per training sample.

PRISM architecture shows advantage in training efficiency when doing context alignment. While a monolithic drafter with a comparable parameter count requires propagating activations and gradients through its entire network depth for each step, both the forward pass and the subsequent backpropagation are confined to a sub-network for PRISM.

For the training of PRISM, we first start with a warm up stage, training the model with just one processing module,

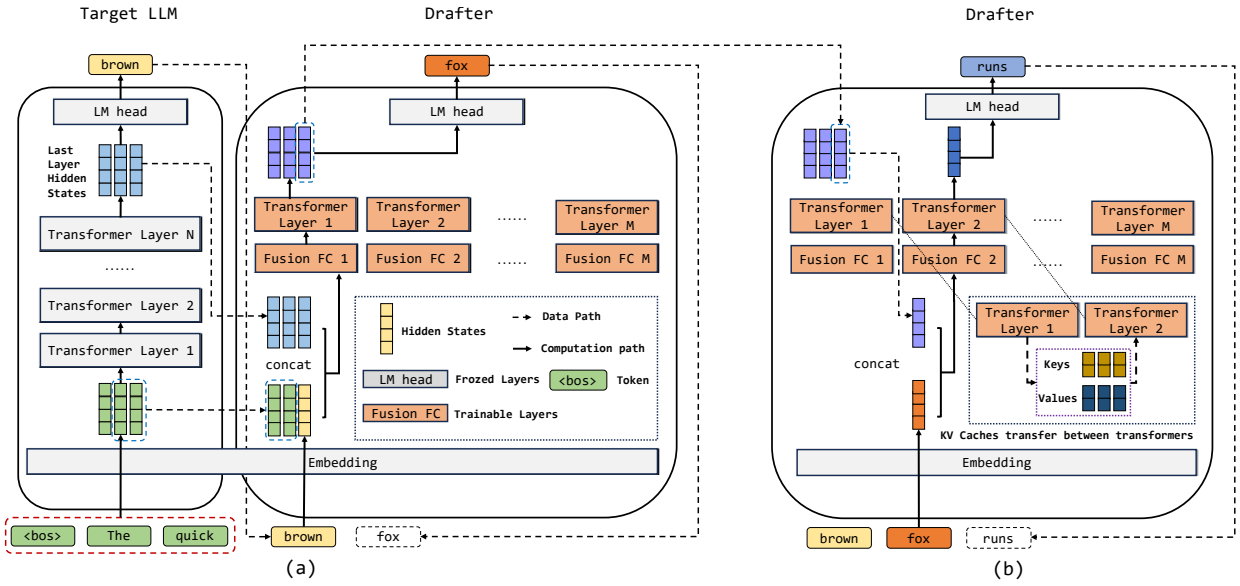

Figure 3. An illustration of the PRISM architecture and its computing paths. (a) The target model prefills or verifies multiple tokens and the output hidden states of its last-layer transformer is used as an input for the draft model, fused with the input token embedding. The fused features is fed to the first transformer of the PRISM drafter for prefill. (b) After fusing the embedding of the previous generated token and the previous generated hidden states, apply the fused feature to the second transformer layer for the first decoding step. Note that KV caches from transfomer 1 is transfered to transformer 2.

meaning that there is no switch of parameters between forward passes. The training loss consists of a Cross Entropy Loss (CEL) comparing draft model output token distributions with ground truth tokens and a Mean Squared Error (MSE) aligning the target model produced hidden states and the draft model produced hidden states.

$$L_1 = \sum_{t=1}^{T} \text{CEL}(\text{softmax}(LM_{head}(z_i^{(t)})), y_i)$$

$$L_2 = \sum_{t=1}^{T} \text{MSE}(z_i^{(t)}, z_i)$$

$$L_{train} = \lambda_1 L_1 + \lambda_2 L_2$$

Where z_i is the i-th target model generated hidden states and $z_i^{(t)}$ is the i-th hidden states outputted by the draft model after its t-th forward pass for the sample. y_i is the i-th ground truth token . λ_1 and λ_2 are coefficients.

After the warm up we replicate the trained transformer weights, producing M copies of the trained transformer and then perform a second phase training. At this time we remove the MSE loss and switch transformers between forward passes.

3.3 Integrating with SGLang

Although implementing the speculative algorithm in PyTorch (Paszke et al., 2019) is convenient, the efficacy of speculative decoding could be overestimated because of missing optimizations from the deployment scenario, such as CUDA graphs and continues batching. Therefore, to concretely evaluate the effectiveness, integrating PRISM with an efficient inference framework like SGLang is necessary. We implement PRISM inside the SGLang engine and orchestrate PRISM with CUDA graph, continues batching, and other inference optimizations.

4 EXPERIMENTS

We experiemented PRISM with a variety of settings.

Target LLMs. We conduct extensive experiments on two widely-used open-source LLMs, i.e., LLaMA-2-chat-7B and LLaMA-3-Instruct-8B (Abbreviated as LLaMA-2-7B and LLaMA-3-8B below). We choose the models because they serve as target LLMs in most previous works. Though sharing the same hidden state size, LLaMA-2-7B and LLaMA-3-8B differ greatly in many aspects. Firstly, LLaMA-3 builds its vocabulary with Byte Pair Encoding (BPE) while LLaMA-2 adopts the SentencePiece, resulting in very different vocabularies; LLaMA-3 employs the Grouped Query Attention (GQA) while LLaMA-2 does not; They are also trained with very different datasets and training pipelines. Draft model is sensitive to all of these factors.

Datasets. To verify scaling laws and evaluate the scaling ability of various draft model architectures, we collect

Table 3. Constitutions of the training data.

Dataset	Volume	Proportion	Topic Coverage
ShareGPT	68K	8.5%	QA/Code/Math/Logic
UltraChat	463K	57.9%	QA
OpenThoughts2	269K	33.6%	Math/Logic

and preprocess data from several widely-used open-source datasets, including ShareGPT, UltraChat (Ding et al., 2023), and OpenThoughts2 (Guha et al., 2025). We preprocess the data by truncating each sample to a maximum length of 2048 tokens. For samples drawn from the OpenThoughts dataset, which is designed to train thinking behavior in language models and contains chain-of-thought reasoning enclosed between explicit opening and closing tags, we strip the thinking content, due to the consideration that our target models, LLaMA-2-Chat-7B and LLaMA-3-Instruct-8B, were not trained with thinking behavior and may not generalize well to such a corpus. Approximately 800,000 samples are drawn from these three datasets. The constitution of the dataset is shown in Table 3. For a fair comparison, the baseline methods mentioned in this paper are also trained on this same dataset.

Benchmarks. Following previous works, we evaluate on six benchmarks,i.e.,MT-Bench (Chen et al., 2025),HumanEval (Chen et al., 2021),GSM8K (Cobbe et al., 2021), Alpaca (Taori et al., 2023), CNN/Daily Mail (See et al., 2017) and Natural Questions (Kwiatkowski et al., 2019), each representing very different workloads. MT-bench is for multi-round conversation; HumanEval is for code completion; GSM8k evaluates LLM on grade-school-level math; Alpaca is a dataset for instrution following; CNN/Daily Mail is for summarization tasks and Natural Questions consists of real questions the users search on google.

Metrics. We focus on the following metrics to evaluate draft models:

- **Acceptance Length**. Acceptance Length is the expected number of draft tokens that the target model accepts per verification step. It reflects how good the draft model predicts target model output.
- **Throughput**. Throughput, measured in tokens per second, measures the end-to-end rate at which a speculative decoding system generates tokens. It is affected not only by the acceptance length but also by the computational overhead.

Baselines. We compare PRISM with various baseline drafter models, including the standard speculative decoding, EAGLE-2, HASS and EAGLE-3.

Train Setting. We train PRISM and all baseline drafters on datasets rangeing from 100,000 to 800,000 samples with 8 NVIDIA A100 40G GPUs and 4 NVIDIA A100 80G GPUs. We use a 95 to 5 split of the data for training and validation, respectively. We optimize using AdamW with $\beta_1 = 0.99$ and $\beta_2 = 0.95$. Gradients are clipped to a maximum norm of 0.5 and a two-stage learning-rate scheduler with linear decay is employed. For the warmup stage: peak learning rate is 3e-5, training for 25–40 epochs depending on the amount of training data. For the tuning stage: peak learning rate is 1e-5, trained for 10 epochs. The training batch size is 1. The time cost for the trainings vary on dataset sizes, ranging from 1 day to 2 weeks.

Note that PRISM, as evaluated in the main experiments, comprises two processing modules: the first is dedicated exclusively to the prefill step, while the second handles all subsequent decoding steps, which is a natural design choice. We further explore PRISM configurations with additional processing modules in Section 4.2.

4.1 Effectiveness

Table 4 shows our main results, comparing PRISM with baseline methods on six workloads, two target models and under greedy and non-greedy sampling settings. The experiments simulate the condition where 1 NVIDIA A800 80G GPU is used for target LLM generation with speculative sampling. An extra set of experiments on 1 NVIDIA H800 80G GPUs can be found in Table 5. All experiments are conducted with batch size 1. A 6-step, 4-branch tree structure is used for the tree-shape draft and verification. A maximum of 16 tokens are validated for each verification. Consistent advantages over all baseline methods in all testing settings is observed. PRISM consistently gains more than 2.4x acceleration compared to the standard vanilla auto-regressive decoding . Compared to other baseline drafter models that activate the same amount of parameters during inference, PRISM on average improves acceptance length by 14.09% and 5.69% and generate 14.21% and 6.10% more tokens per second than EAGLE-2 and HASS, respectively. Speculative decoding performance is task dependent. PRISM works exceptionally well, compared to other baselines on CNN/-Daily Mail work load, outperforms HASS by potentially suggests thGPUe step splitting architecture may work better for long context tasks. In this experiment, running PRISM with SGLang requires an additional 576 MB of GPU memory compared to one-transformer-layer draft model methods (EAGLE-2, HASS). Relative to the overall SGLang memory footprint, this corresponds to only a 0.77% increase.

4.2 Re-examine Scaling Laws of PRISM

We conduct experiments to verify the drafter scaling phenomenon observed in previous works (Li et al., 2025a; Yan et al., 2025). We also prove with these experiments that even without increasing the number of activated parameters

Table 4. Acceptance lengths (AL) and throughput (TPS, tokens per second) of different methods on NVIDIA A800 GPU.

			MT-bench		HumanEval		GSM8K		Alpaca		CNN/DM		Natural Ques.	
Model	Method	Temp	AL	TPS	AL	TPS	AL	TPS	AL	TPS	AL	TPS	AL	TPS
LLaMA-2	Vanilla*	T = 0	N/A	95.66	N/A	96.10	N/A	96.51	N/A	96.75	N/A	92.77	N/A	96.62
		T = 1	N/A	95.70	N/A	96.08	N/A	96.31	N/A	96.51	N/A	92.66	N/A	96.447
	Standard**	T = 0	2.68	142.35	2.70	145.65	2.88	155.23	2.88	155.49	2.17	109.92	2.83	152.58
		T = 1	2.68	142.37	2.55	136.65	2.77	146.76	2.58	137.17	2.12	104.46	2.61	138.58
	EAGLE-2	T = 0	4.16	246.18	4.78	283.84	4.29	254.13	4.08	243.54	3.88	218.83	3.80	225.49
		T = 1	3.87	223.69	4.49	260.25	4.26	246.24	3.87	225.22	3.70	210.09	3.60	208.45
	HASS	T = 0	4.48	265.08	5.15	305.31	4.54	268.94	4.42	263.70	4.18	235.41	4.08	241.61
		T = 1	4.12	238.13	4.77	276.39	4.43	256.26	4.21	244.44	3.99	226.91	3.79	219.49
	PRISM (ours)	T = 0	**4.66**	**274.32**	**5.33**	**316.16**	**4.73**	**279.14**	**4.67**	**277.35**	**4.40**	**255.08**	**4.24**	**250.31**
		T = 1	**4.30**	**246.64**	**4.90**	**284.43**	**4.68**	**269.19**	**4.39**	**254.15**	**4.15**	**231.13**	**4.02**	**229.80**
LLaMA-3	Vanilla*	T = 0	N/A	85.63	N/A	85.88	N/A	85.98	N/A	86.19	N/A	83.18	N/A	86.12
		T = 1	N/A	85.39	N/A	85.70	N/A	85.85	N/A	85.79	N/A	82.90	N/A	85.86
	Standard**	T = 0	5.06	164.13	5.77	180.66	5.47	173.28	4.56	144.51	4.43	143.18	4.30	136.81
		T = 1	4.42	142.69	5.45	169.80	5.20	163.19	4.16	131.12	4.17	124.41	3.77	119.45
	EAGLE-2	T = 0	3.67	168.26	4.57	212.36	4.24	197.66	3.67	171.89	3.59	163.34	3.14	147.74
		T = 1	3.67	168.64	4.30	182.15	4.00	172.00	3.37	146.32	3.30	137.69	2.99	129.65
	HASS	T = 0	3.93	180.50	5.12	235.38	4.73	216.75	3.94	184.39	3.90	176.16	3.36	156.67
		T = 1	3.93	181.14	4.84	206.69	4.49	191.51	3.62	155.73	3.58	147.01	3.08	133.46
	PRISM (ours)	T = 0	**4.29**	**201.51**	**5.43**	**251.96**	**5.11**	**237.41**	**4.25**	**199.31**	**4.17**	**187.95**	**3.63**	**170.06**
		T = 1	**4.29**	**201.47**	**5.02**	**216.30**	**4.76**	**205.09**	**3.86**	**167.72**	**3.76**	**158.33**	**3.31**	**143.63**

* Vanilla means the standard auto-regressive decoding. Thus, there is no acceptance length for vanilla decoding.
** Standard means using a light weight version of the same series models as the drafter. For LLaMA-2-7B we employ the LLaMA-160M model while for LLaMA-3-8B we employ the LLaMA-3.1-1B model. Same with the experiments on NVIDIA H800 GPUs.

per prediction, the model's capacity expands as the volume of training data grows. This expansion is evidenced by a clear improvement in predictive performance. To more clearly demonstrate the upper bound of predictive power for various draft models at different training data scales, we employed a more extreme tree structure in the experiments: a 6-step, 10-branch tree, with each step verifying up to 60 tokens. Results for a more realistic tree structure used during generation are provided in Appendix B.

The first results are presented in Figure 4, comparing PRISM with drafters that activate a similar amount of parameters during inference, i.e., EAGLE-2 and HASS. Across all tested data scales, PRISM demonstrates a big advantage in acceptance length over both EAGLE-2 and HASS.

We further analyze the curves. First, in the low-data regime with fewer than 200,000 training samples, the representational capacity of the single-transformer models has not yet been saturated. This is indicated by their performance continuing to improve with more data. Nevertheless, PRISM already outperforms both baselines at this scale. This supports our hypothesis that the hierarchical structure of PRISM progressively generates more informative representations as data passes through its stacked processing modules, leading to better predictions. Second, in the high-data regime, corresponding to the latter half of the curves, a clear scaling bottleneck becomes evident for both EAGLE-2 and HASS. While PRISM still shows potential to scale with more than 600,000 samples, the performance of EAGLE-2 and HASS begins to plateau after 400,000 samples, respectively. This demonstrates the superior scaling ability of PRISM compared to single-transformer draft models like EAGLE-2 and HASS.

We also compare PRISM against EAGLE-3, an architecture specifically designed for scaling. EAGLE-3 expands attention layer parameters and leverages multiple target LLM produced hidden states, all of which PRISM does not equip.

Figure 5 compares the average scaling ablity of PRISM and EAGLE-3, when trained as drafters for LLaMA-3-8B. PRISM demonstrates scaling behavior comparable to EAGLE-3 without the multi-hidden-state input feature, and fully surpasses EAGLE-3 when this feature is incorporated (denoted as PRISM*).

We further conduct an ablation study to verify that the scaling efficacy of PRISM stems from its core architectural innovation, which is distributing draft-step computation across distinct sets of parameters, rather than from merely increasing parameter count. Specifically, we compare PRISM against HASS-style baselines that naively stack multiple transformer layers as the draft head. We train a two-layer variant (HASS-2) and a three-layer variant (HASS-3). The results are shown in Figure 6. On one hand, HASS-3 consistently outperforms HASS-2 across all data volumes and temperature settings, confirming the parameter-scaling findings

Table 5. Acceptance lengths (AL) and throughput (TPS, tokens per second) of different methods on NVIDIA H800 GPU.

			MT-bench		HumanEval		GSM8K		Alpaca		CNN/DM		Natural Ques.	
Model	Method	Temp	AL	TPS	AL	TPS	AL	TPS	AL	TPS	AL	TPS	AL	TPS
LLaMA-2	Vanilla	T = 0	N/A	142.10	N/A	142.37	N/A	143.08	N/A	143.43	N/A	139.76	N/A	142.99
		T = 1	N/A	142.12	N/A	140.76	N/A	141.70	N/A	141.22	N/A	138.42	N/A	141.47
	Standard	T = 0	2.73	188.02	2.76	193.47	2.91	202.73	2.90	204.16	2.06	141.69	2.87	200.25
		T = 1	2.74	191.44	2.60	180.92	2.82	195.00	2.85	198.09	2.05	139.84	2.75	190.07
	EAGLE-2	T = 0	4.12	335.71	4.72	387.42	4.28	347.95	4.07	333.99	3.86	309.78	3.81	309.43
		T = 1	4.12	337.17	4.37	319.90	4.14	298.77	3.90	282.88	3.69	263.54	3.54	257.02
	HASS	T = 0	4.40	349.75	5.08	408.49	4.54	360.79	4.38	350.96	4.17	327.60	4.05	321.47
		T = 1	4.38	350.88	4.77	342.02	4.38	310.17	4.06	290.11	3.98	279.20	3.85	271.97
	PRISM (ours)	T = 0	**4.67**	**373.94**	**5.36**	**432.93**	**4.77**	**381.51**	**4.68**	**377.50**	**4.40**	**345.78**	**4.32**	**343.94**
		T = 1	**4.68**	**377.55**	**4.82**	**348.27**	**4.68**	**333.45**	**4.40**	**316.14**	**4.21**	**297.57**	**3.97**	**283.70**
LLaMA-3	Vanilla	T = 0	N/A	145.10	N/A	145.34	N/A	145.52	N/A	145.89	N/A	144.03	N/A	145.64
		T = 1	N/A	146.03	N/A	144.10	N/A	144.09	N/A	144.20	N/A	143.32	N/A	143.80
	Standard	T = 0	5.07	241.19	5.89	276.13	5.75	271.18	4.80	228.80	4.72	221.31	4.39	209.10
		T = 1	5.05	241.55	5.41	254.10	5.35	250.34	4.22	199.23	4.21	197.30	3.88	182.53
	EAGLE-2	T = 0	3.72	202.10	4.58	312.94	4.27	291.35	3.67	253.10	3.57	243.60	3.25	223.44
		T = 1	3.72	257.92	4.31	210.01	3.98	192.28	3.41	165.85	3.32	161.04	2.99	146.24
	HASS	T = 0	4.00	278.70	5.08	349.65	4.75	326.78	3.93	273.59	3.91	268.37	3.40	236.46
		T = 1	4.00	280.60	4.81	235.42	4.51	220.36	3.64	179.08	3.55	174.53	3.08	152.45
	PRISM (ours)	T = 0	**4.63**	**315.42**	**5.83**	**392.05**	**5.51**	**369.40**	**4.53**	**308.54**	**4.60**	**309.17**	**3.91**	**266.71**
		T = 1	**4.62**	**316.13**	**5.30**	**255.74**	**5.03**	**241.16**	**4.03**	**195.39**	**4.08**	**196.56**	**3.51**	**170.65**

reported in prior work (Yan et al., 2025; Tang et al., 2025): adding more parameters to the draft head does improve acceptance length. On the other hand, PRISM surpasses both HASS-2 and HASS-3 by a clear margin despite using a comparable or less parameter budget. This gap demonstrates that the performance gains of PRISM cannot be attributed to increased capacity alone. Rather, by factoring the multi-step drafting computation into step-specific parameter sets, PRISM allocates its capacity more efficiently than a monolithic stack of transformer layers, yielding stronger scaling behavior with the same training resources.

In the experiments above, we employ a PRISM instance comprising two processing modules, dedicated to the prefill step and all subsequent steps, respectively. As our formal definition of the PRISM architecture permits, one can also employ more processing modules with more flexible parameter mappings. We therefore investigate an alternative draft-step-to-processing-module assignment by extending PRISM with an additional processing module. As illustrated in Figure 7, we refer to the two-module configuration as PRISM-2, and to the three-module configuration as PRISM-3. The latter retains the same dedicated prefill module but assigns each of the two subsequent draft steps its own independent processing module. As shown in Figure 7, PRISM-3 holds a slight advantage at small data volumes, as the step-specific modules provide extra capacity by specializing in their respective draft positions. However, as data scales up, PRISM-2 steadily overtakes PRISM-3 and ultimately achieves notably higher acceptance length. Although PRISM-3 appears to underperform at larger data volumes, this does not necessarily indicate that adding more processing modules is ineffective. By assigning each draft step its own module, PRISM-3 halves the training signal available to each of the two step-specific modules compared with the shared module in PRISM-2. Moreover, the training data for the later draft steps tends to be noisier, as prediction difficulty compounds with distance from the base model's last hidden state. The combination of reduced data volume and increased noise likely leaves the step-specific modules in an under-trained regime, a phenomenon analogous to the well-known issue in MoE architectures where individual experts fail to be fully trained due to insufficient routing frequency. Indeed, the early-stage advantage of PRISM-3 at 100k suggests that its additional capacity is genuinely beneficial; the subsequent plateau and regression likely reflect an optimization challenge rather than a fundamental architectural limitation.

4.3 Hyper-Parameters

Speculative decoding is a complex system where lots of hyper parameters influences the end-to-end acceleration. In this section, we demonstrate these influences, revealing insights for achieving higher end-to-end throughput.

Batch Size. The benefit of speculative sampling diminishes with increasing batch sizes, as larger batch sizes make LLM decoding more compute-intensive and less constrained by memory bandwidth. It is often a trade-off between employing speculative sampling and utilizing big batch sizes. Nonetheless, PRISM still provides reasonable speedup when

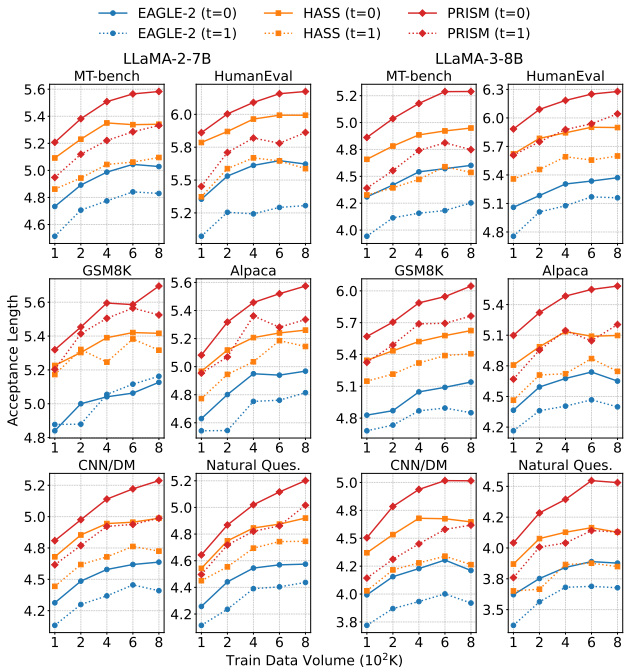

Figure 4. Scaling curves of PRISM and comparative drafters over 6 benchmarks and greedy and non-greedy sampling.

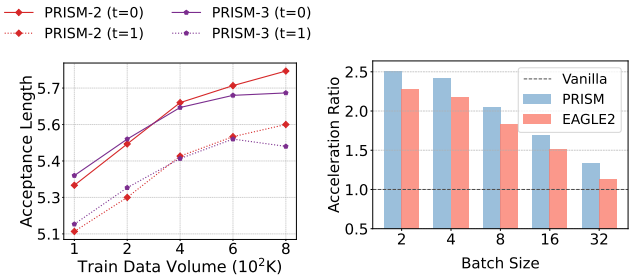

Figure 7. PRISM-2 vs. PRISM-3.

Figure 8. A comparision between PRISM and EAGLE-2 acceleration with larger batch size.

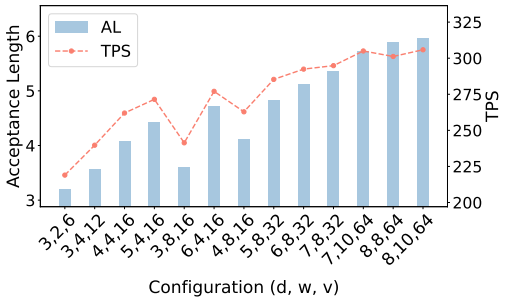

Figure 9. PRISM acceptance length and throughput under different tree topologies.

evaluated with relatively larger batch sizes, as shown in Figure 8.

Tree Topology. In speculative sampling, the tree topology defines the depth (d) and width (w) of the tree of candidate tokens generated by the draft model. To keep the computational cost of the validation step manageable, the total number of verified nodes (v) is typically fixed. The optimal tree topology, however, is highly dependent on the predictive characteristics of the draft model. For instance, a draft model capable of accurate long-range predictions may perform best with a deeper, narrower tree. Conversely, a model whose predictive accuracy drops off more quickly may benefit from a wider, shallower tree that explores more parallel branches. In Figure 9, we explore the effect tree topology has on PRISM.

Experimental results indicate that employing a larger tree generally enhances the performance of PRISM. However, speedups improve with growing tree depth and width gradually saturate. This is due to the growing overhead of both draft and verify stages coming with larger trees. Specifically, for a fixed number of tokens to be verified, a deeper, narrower tree structure outperforms a shallower, wider one.

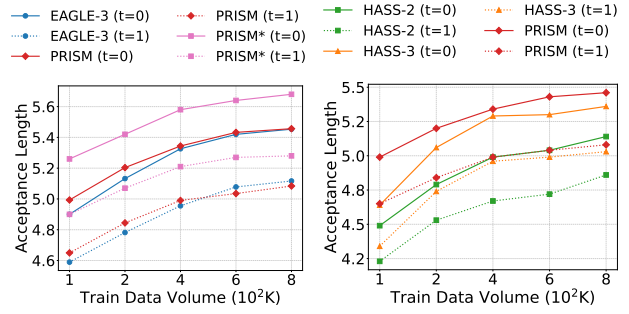

Figure 5. PRISM vs. EAGLE-3.

Figure 6. Ablation for PRISM.

5 Related Works

Apply speculative sampling techniques to accelerate LLMs was first introduced by Leviathan et al. 2023. Since that, extensive efforts have been made from both the system and AI research community to improve the framework.

Efforts have been made exploring speculative sampling in LLM serving systems. SpecInfer (Miao et al., 2024) firstly introduce the tree-based speculative inference and token verification mechanism instead of an incremental decoder, which significantly reduces the verification overhead. SwiftSpec (Zhang et al., 2025b) finds the imbalanced compute requirements for draft and target model under the common circumstance that the draft and target model use the same tensor parallel configuration. It redesigns the speculative

decoding pipeline in an asynchronous and disaggregated method, thus assign the computation resources by requirements. AdaServe (Li et al., 2025b) supports multi-SLO requirements through SLO-customized speculative decoding. It improves the goodput and reduces SLO violations by the hardware-aware algorithm and speculate-select-verify pipeline. Besides, because reinforcement learning including rollout becomes a pivotal methodology for enhancing LLMs and imbalances in rollout lengths cause low GPU utilization, speculative sampling is also a promising method to alleviate this problem and accelerate RL training (He et al., 2025).

Another significant body of work has focused on improving the efficiency of speculative decoding by increasing the number of accepted tokens per step. Medusa (Cai et al., 2024) pioneered the multi-head approach for draft generation. Instead of using a separate draft model, Medusa augments a pretrained backbone by attaching multiple, lightweight decoding heads to its final layer. The EAGLE series (Li et al., 2024a;b; 2025a) further improves acceptace length by feeding richer feature representations from the target model into the draft heads. This enhanced context leads to higher acceptance rates and, consequently, greater end-to-end speedups. EAGLE-2 introduced a dynamic tree-building algorithm, while EAGLE-3 further improved performance by incorporating a context alignment technique during training and leveraging additional hidden states from the target model. HASS (Zhang et al., 2025a) independently developed its own version of context alignment, proposing a novel two-phase training scheme and a distinct loss function to optimize. Focusing on the same challenge of context mismatch, Griffin (Hu et al., 2025) proposed that not only hidden states but also the input tokens themselves should be synchronized between the draft and target models during training and inference. Shifting the focus, Jakiro (Huang et al., 2025), notable for its performance in non-greedy sampling scenarios, introduced MOE and contrastive learning into the draft model training.

Finally, the challenge of scaling draft models was explored by Scylla (Yan et al., 2025), which investigated the impact of increasing the parameter count of the drafters. A concurrent work from Meta (Tang et al., 2025) independently conducted similar experiments, confirming that larger, more capable drafters can yield substantial performance gains.

6 Conclusion and Future Work

In this paper, we introduce PRISM, a novel architecture for draft model of speculative sampling. PRISM deals with the central trade-off between draft model effectiveness and efficiency. By parametrically disaggregating the computation of draft steps, draft model capacity increases while computation cost for each step maintains. Extensive experiments are performed, verifying scaling law for draft models while proving effective scaling for the PRISM architecture. We also demonstrate the practicability of PRISM in real-life scenarios by integrating it into sglang and proving the effectiveness.

Due to the constraints of our current hardware platform, we were unable to validate the effectiveness of PRISM with larger target models such as LLaMA-2-70B, nor to verify scaling behavior on significantly larger training data regimes, as these experiments demand substantial GPU memory, high-bandwidth inter-node interconnects, and greater overall compute budgets. We plan to extend our experiments to these larger-scale configurations in future work. We believe that exploring these settings would further strengthen the empirical evidence for the scaling properties of PRISM and its practical benefits in production-grade serving systems.

7 Acknowledgements

We appreciate the constructive comments of the reviewers. This work is supported by funding from CUHK (4937007, 4937008, 5501329, 5501517), Canada's NSERC OGP0046506, and Canada Research Chair Program.

References

Butler, B., Yu, S., Mazaheri, A., and Jannesari, A. Pipeinfer: Accelerating llm inference using asynchronous pipelined speculation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '24. IEEE Press, 2024. ISBN 9798350352917. doi: 10.1109/SC41406.2024.00046. URL https://doi.org/10.1109/SC41406.2024.00046.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Chen, J., Feng, A., Zhao, Z., Garza, J., Nurbek, G., Qin, C., Maatouk, A., Tassiulas, L., Gao, Y., and Ying, R. Mtbench: A multimodal time series benchmark for temporal reasoning and question answering, 2025. URL https://arxiv.org/abs/2503.16858.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov,

M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Ding, N., Chen, Y., Xu, B., Qin, Y., Zheng, Z., Hu, S., Liu, Z., Sun, M., and Zhou, B. Enhancing chat language models by scaling high-quality instructional conversations, 2023. URL https://arxiv.org/abs/2305.14233.

Google, G. T. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Guha, E., Marten, R., Keh, S., Raoof, N., Smyrnis, G., Bansal, H., Nezhurina, M., Mercat, J., Vu, T., Sprague, Z., Suvarna, A., Feuer, B., Chen, L., Khan, Z., Frankel, E., Grover, S., Choi, C., Muennighoff, N., Su, S., Zhao, W., Yang, J., Pimpalgaonkar, S., Sharma, K., Ji, C. C.-J., Deng, Y., Pratt, S., Ramanujan, V., Saad-Falcon, J., Li, J., Dave, A., Albalak, A., Arora, K., Wulfe, B., Hegde, C., Durrett, G., Oh, S., Bansal, M., Gabriel, S., Grover, A., Chang, K.-W., Shankar, V., Gokaslan, A., Merrill, M. A., Hashimoto, T., Choi, Y., Jitsev, J., Heckel, R., Sathiamoorthy, M., Dimakis, A. G., and Schmidt, L. Openthoughts: Data recipes for reasoning models, 2025. URL https://arxiv.org/abs/2506.04178.

He, J., Li, T., Feng, E., Du, D., Liu, Q., Liu, T., Xia, Y., and Chen, H. History rhymes: Accelerating llm reinforcement learning with rhymerl, 2025. URL https://arxiv.org/abs/2508.18588.

Hu, S., Li, J., Xie, X., Lu, Z., Toh, K.-C., and Zhou, P. Griffin: Effective token alignment for faster speculative decoding, 2025. URL https://arxiv.org/abs/2502.11018.

Huang, H., Yang, F., Liu, Z., Xu, Y., Li, J., Liu, Y., Yin, X., Li, D., Ren, P., and Barsoum, E. Jakiro: Boosting speculative decoding with decoupled multi-head via moe, 2025. URL https://arxiv.org/abs/2502.06282.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Devlin, J., Lee, K., Toutanova, K., Jones, L., Kelcey, M., Chang, M.-W., Dai, A. M., Uszkoreit, J., Le, Q., and Petrov, S. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23)*, pp. 61–626, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle: Speculative sampling requires rethinking feature uncertainty, 2024a. URL https://arxiv.org/abs/2401.15077.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-2: Faster inference of language models with dynamic draft trees, 2024b. URL https://arxiv.org/abs/2406.16858.

Li, Y., Wei, F., Zhang, C., and Zhang, H. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025a. URL https://arxiv.org/abs/2503.01840.

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., and Stoica, I. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679, Boston, MA, July 2023. USENIX Association. ISBN 978-1-939133-34-2.

URL https://www.usenix.org/conference/osdi23/presentation/li-zhouhan.

Li, Z., Chen, Z., Delacourt, R., Oliaro, G., Wang, Z., Chen, Q., Lin, S., Yang, A., Zhang, Z., Chen, Z., Lai, S., Cheng, X., Miao, X., and Jia, Z. Adaserve: Accelerating multi-slo llm serving with slo-customized speculative decoding, 2025b. URL https://arxiv.org/abs/2501.12162.

Masson, D., Malacria, S., Casiez, G., and Vogel, D. Directgpt: A direct manipulation interface to interact with large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642462. URL https://doi.org/10.1145/3613904.3642462.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., Shi, C., Chen, Z., Arfeen, D., Abhyankar, R., and Jia, Z. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

OpenAI. Introducing chatgpt. https://openai.com/index/chatgpt/, November 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I. n., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ISCA '24, pp. 118–132. IEEE Press, 2025. ISBN 9798350326581. doi: 10.1109/ISCA59077.2024.00019. URL https://doi.org/10.1109/ISCA59077.2024.00019.

Qin, R., Li, Z., He, W., Cui, J., Ren, F., Zhang, M., Wu, Y., Zheng, W., and Xu, X. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170, Santa Clara, CA, February 2025. USENIX Association. ISBN 978-1-939133-45-8. URL https://www.usenix.org/conference/fast25/presentation/qin.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL https://www.aclweb.org/anthology/P17-1099.

Tang, B., Fu, C. C., Kou, F., Sizov, G., Zhang, H., Park, J., Liu, J., You, J., Yang, Q., Mehta, S., Cai, S., Wang, X., Liu, X., Li, Y., Zhou, Y., Wei, W., Zhao, Z., Qi, Z., Victoria, A., Ibrahim, A., Wasti, B., Kim, C., Haziza, D., Sun, F., Delfin, G., Guo, E., Ouyang, J., Lee, J., Huang, J., Reizenstein, J., Fang, L., Zhu, Q., Verma, R., Mihailescu, V., Guo, X., Cui, Y., Hu, Y., and Lee, Y. Efficient speculative decoding for llama at scale: Challenges and solutions, 2025. URL https://arxiv.org/abs/2508.08192.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Xia, Y., Shenoy, M., Jazdi, N., and Weyrich, M. Towards autonomous system: flexible modular production system enhanced with large language model agents. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2023. doi: 10.1109/ETFA54631.2023.10275362.

Yan, S., Zhu, M., qing Jiang, G., Wang, J., Chen, J., Zhang, W., Liao, X., Cui, X., Zhang, C., Song, Z., and Zhu, R. Scaling laws for speculative decoding, 2025. URL https://arxiv.org/abs/2505.07858.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Zhang, L., Wang, X., Huang, Y., and Xu, R. Learning harmonized representations for speculative sampling. In *International Conference on Learning Representations*, 2025a.

Zhang, Z., Jiang, Z., Jiang, C., Yu, M., Zheng, S., Lin, H., Hoffmann, H., and Liu, X. Swiftspec: Ultra-low latency llm decoding by scaling asynchronous speculative decoding, 2025b. URL https://arxiv.org/abs/2506.11309.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24, USA, 2024a. USENIX Association. ISBN 978-1-939133-40-3.

Zhong, Y., Yang, Z., Li, Z., Chen, R., Yu, Z., Zheng, L., Ho, Q., Gonzalez, J. E., Stoica, I., Zhang, H., and Jin, X. TetriInfer: A case for GPU inference service with elastic cold storage. In *21st USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 907–925, Santa Clara, CA, July 2024b. USENIX Association. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.