Steering Pretrained Drafters during Speculative Decoding

Frédéric Berdoz
ETH Zürich
fberdoz@ethz.ch

Peer Rheinboldt
ETH Zürich
prheinboldt@ethz.ch

Roger Wattenhofer
ETH Zürich
wattenhofer@ethz.ch

Abstract

Speculative decoding accelerates language model inference by separating generation into fast drafting and parallel verification. Its main limitation is drafter–verifier misalignment, which limits token acceptance and reduces overall effectiveness. While small drafting heads trained from scratch compensate with speed, they struggle when verification dominates latency or when inputs are out of distribution. In contrast, pretrained drafters, though slower, achieve higher acceptance rates thanks to stronger standalone generation capabilities, making them competitive when drafting latency is negligible relative to verification or communication overhead. In this work, we aim to improve the acceptance rates of pretrained drafters by introducing a lightweight dynamic alignment mechanism: a *steering vector* computed from the verifier's hidden states and injected into the pretrained drafter. Compared to existing offline alignment methods such as distillation, our approach boosts the number of accepted tokens by up to 35% under standard sampling and 22% under greedy sampling, all while incurring negligible computational overhead. Importantly, our approach can be retrofitted to existing architectures and pretrained models, enabling rapid adoption.

1 Introduction

The auto-regressive nature of transformer-based large language models (LLMs) [44] inherently limits their inference speed. This limitation is further amplified by the rapid growth in model size among frontier LLMs [1, 15, 25, 46]. Numerous approaches have been proposed to reduce latency, including weight quantization [9], model pruning [16], and distillation [17], but these often come at the expense of generated text quality. A paradigm that escapes this trade-off is *speculative decoding* [21, 45, 6], which follows the general principle of *speculative execution* [3]. This method employs a lightweight *drafter* to propose the next k tokens, which are then verified in parallel using a single forward pass of the larger base model, commonly referred to as the *verifier*. In essence, speculative decoding leverages the underutilization of accelerator hardware in classic auto-regressive decoding by using batched verification to amortize the costly transfer of model parameters between off-chip memory and on-chip cache. Two main families of approaches have emerged for speculative decoding [18]. The first uses an independent drafter [45], typically a compact LLM trained independently on similar data as the verifier. Since these drafters are capable language models in their own right, they can generalize reasonably well, even without task-specific tuning or dynamic steering. The second family of approach employs small dependent speculative heads mounted directly on top of the verifier and trained from scratch [39, 4, 2, 22]. At inference, these methods rely mostly on dynamic steering to keep the drafter aligned with the verifier despite its limited capacity. Although such drafters often produce shorter accepted blocks, their low latency allows them to rapidly generate many candidate sequences. Combined with efficient batch evaluation [30], this makes them competitive in settings where the cost of verification is relatively low, such as in controlled research environments. However,

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Structured Probabilistic Inference & Generative Modeling.

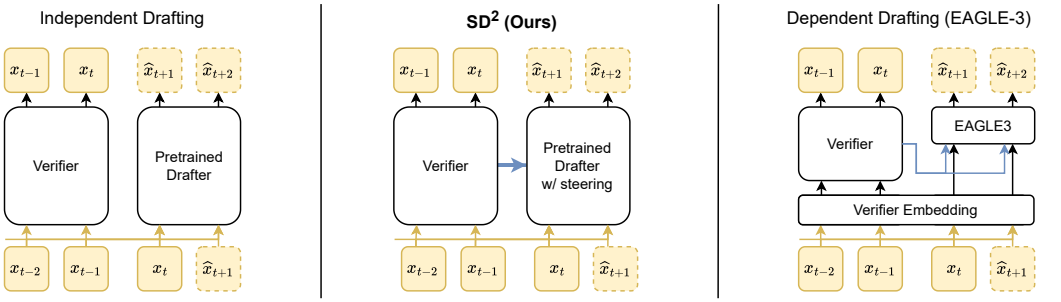

Figure 1: Overview of different drafting paradigms: Independent drafting uses a smaller model from the same family as the verifier, with no access to its internal state. Dependent drafting (e.g., EAGLE-3) uses lightweight heads trained to read the verifier's hidden states, sharing input embeddings and using concatenated features for guidance. SD^2 strikes a middle ground, leveraging verifier features for steering while retaining the generalization capabilities of independent drafters.

in real-world scenarios where verification latency fluctuates or dominates total runtime, e.g., when the verifier is remote [32], deployed on slower hardware, shared across several drafters, or simply frontier-scale with 600B+ parameters [25], the block efficiency becomes the key driver of efficacy, as it dictates the number of verification steps. While independent drafters tend to perform better in that regard, due to their ability to generate coherent sequences, they can only rely on their offline alignment with the verifier. Building on the observation that LLMs (verifiers in our case) implicitly encode information about upcoming tokens in their intermediate representations [35], we propose **S**teering pretrained **D**rafters during **S**peculative **D**ecoding (SD^2), a lightweight guiding mechanism that extracts this latent signal to dynamically steer drafters at inference.

Our key contributions include:

- We introduce a lightweight dynamic steering mechanism for pretrained drafters during speculative decoding.
- We show that our steering mechanism improves the number of drafted tokens accepted by up to 35% and has up to 22% higher throughput compared to independent drafters across a variety of tasks and models.
- We motivate our design choices with several ablations.

2 Related Work

2.1 Speculative Decoding

Speculative decoding (SD) originates from the speculative execution paradigm [3]. While early variants only supported greedy decoding acceleration [39, 40, 14, 45], the concurrent works of Leviathan et al. [21] and Chen et al. [6] introduced *speculative sampling*, extending speculative decoding to non-deterministic decoding algorithms. This sparked a long line of work focused on improving the efficiency of such methods, typically evaluated by token throughput (wall-clock speedup) in controlled environments. We refer to Hu et al. [18] for a comprehensive survey and detailed taxonomy of speculative decoding.

Dependent Drafters. The first drafters consisted of several decoding heads that independently drafted tokens to form a sequence, taking the verifier's last hidden state as input [39, 4]. While fast (thanks to parallel token drafting), these methods suffer from the lack of dependency between the drafted tokens, strongly limiting the token acceptance rate. Recognizing this limitation, Li et al. [22] propose to use auto-regressive drafters on the hidden states, and Ankner et al. [2] improves by taking the embeddings of the previously drafted tokens as input to the autoregressive drafter. Instead of only using the last hidden representations of the drafter, Zimmer et al. [51] and Du et al. [11] use the KV values of the verifiers during drafting. Zhang et al. [48] and Li et al. [24] further improve the acceptance rates by training the drafter to use its hidden features to close the gap between training

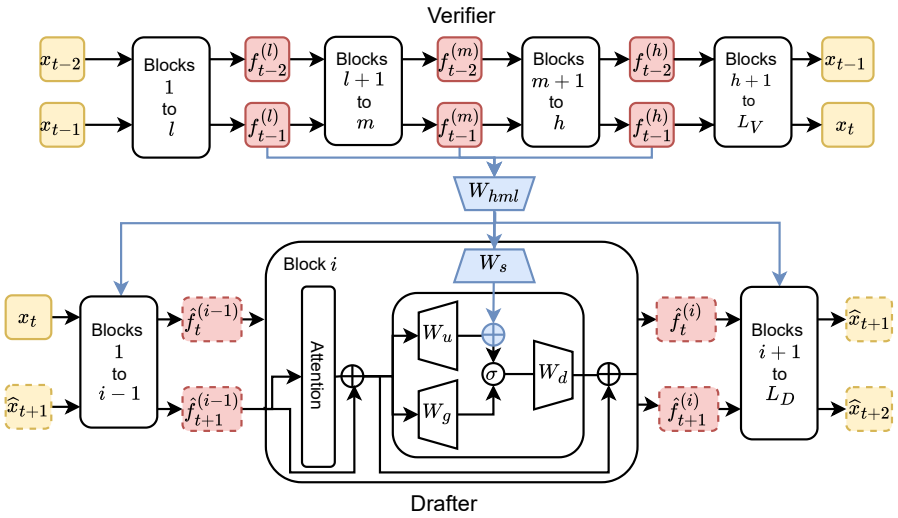

Figure 2: The steering mechanism in SD^2 works by concatenating the verifier's high-, medium-, and low-level hidden features and passing them through a linear projection to produce a steering vector. This embedding is transformed by another linear layer into a set of biases, which are added to all MLP hidden states in the drafter just before the activation function, as detailed in Eq. 1 and Eq. 2.

and inference. Although our study centers on independent drafters, we also report the block efficiency of EAGLE-3 [24] in a chain decoding setting (i.e., only one proposed sequence, excluding its tree decoding component) to provide a reference point for the improvements achievable by independent drafters.

Independent Drafters. Independent auto-regressive drafters were first introduced by Xia et al. [45]. Building on this, Huang et al. [19] proposed an enhanced version where the candidate length is determined on the fly via an acceptance prediction head. Zhou et al. [50] note that the acceptance rate of the drafted token is theoretically bounded by the divergence between the drafter and verifier, and therefore propose to distill the verifier into the drafter. Alternatively, [26] propose online speculative decoding, where drafters are continuously retrained on new user inputs, and Fu et al. [13] propose a drafter-free version using intermediate Jacobi iterations as drafted sequences.

Verification. Sun et al. [41, 42] frame the verification phase as an optimal transport problem to improve batch and block verification, respectively. Spector and Re [38] and Miao et al. [30] introduce tree-based speculative inference, where many drafted sequences are arranged in a tree and verified in parallel. Building on this idea, Li et al. [23] introduce dynamic drafting trees. Lastly, [47] explore the theoretical limits of speculative decoding.

2.2 Dynamic Steering of LLMs

The technique of activation steering, first proposed by Turner et al. [43], allows for the control of LLM behavior by directly modifying model activations during inference. It is primarily motivated by the *linear representation hypothesis* [33], suggesting that a model's intent or behavior is encoded along specific, steerable directions. Subsequently, Rimsky et al. [34] introduced a method to compute steering vectors by averaging the activation differences between sets of positive and negative examples. More recently, Chalnev et al. [5] introduce a method to predict a steering vector's impact on internal sparse autoencoder (SAE) features [20]. However, these approaches focus on static, interpre steering and remain largely unexplored in the dynamic context of speculative decoding.

3 Methodology

Steered speculative decoding (SD^2) follows the standard speculative decoding paradigm of drafting a candidate sequence and verifying each token in parallel [21, 6]. The key addition is that, in addition

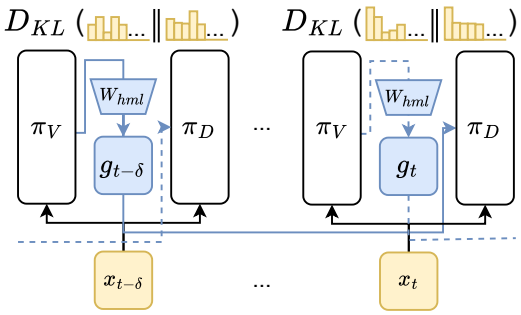

Figure 3: The training process of SD^2 aligns the drafter's (π_D) probability distribution to the verifier's (π_V). To achieve this, we randomly choose an offset $\delta \in [1, k]$ to simulate drafting the δ'th token of a block. After extracting g from on the verifier's activations, we compute $\pi_D(x_t|x_{1:t-1}, g_{t-\delta})$ and use the Kullback-Leibler divergence $D_{\text{KL}}(\pi_V(\cdot|x_{1:t-1})\|\pi_D(\cdot|x_{1:t-1}, g_{t-\delta}))$ as loss. In addition to W_s (see Figure 2), both W_{hml} and π_D are trained. The verifier π_V stays frozen throughout training.

to the rejection of candidate tokens, the verification step also produces a *steering vector*, which is used to guide the drafter in the next generation phase. Our method is motivated by the observation that auto-regressive models implicitly encode information about future tokens beyond the immediate next token, even without being explicitly trained to do so [35]. We aim to extract this predictive information from the verifier's hidden representations and inject it into the drafter to dynamically guide generation. Ablations justifying this mechanism can be found in Appendix A.4.

3.1 Verification

In SD^2, the verification of candidate tokens remains unchanged to the regular speculative decoding framework, where we compute $\pi_V(\hat{x}_{t+i} \mid x_{1:t}, \hat{x}_{t+1:t+i-1})$ for all $i \in [1, k]$ in parallel, and then compare with the drafter's predicted outputs to accept or reject the token using rejection sampling, which has been proven to be optimal [21, 47]. We further enhance this step with the generation of a *steering vector* g_t to further condition π_D. This steering vector is generated based on the verifier's hidden states at the position of the first rejected token, i.e, the last token returned. Similar to EAGLE-3 [24] we use a linear layer, which is applied on the concatenation of h_t, m_t, l_t, which are the high, middle and low activations of the verifier from three different layers, to generate steering vector $g_t = W_{hml}[h_t, m_t, l_t]^\top$.

3.2 Drafting

The drafting process of candidate tokens follows the standard auto-regressive decoding of regular LLMs, except that in SD^2 the probability distributions $\pi_D(\hat{x}_{t+i} \mid x_{1:t}, g_t, \hat{x}_{t+1:t+i-1})$ is further conditioned on the steering vector g_t. To steer the drafter, we incorporate a linear mapping of g_t as a bias in all MLP layers $l = 1, ..., L_D$ of π_D by changing the SwiGLU [37] computation from

$$a_{t+i}^{(l)} \mapsto W_d(W_u a_{t+i}^{(l)} \odot \sigma(W_g a_{t+i}^{(l)})), \tag{1}$$

to

$$a_{t+i}^{(l)}, \mathbf{g_t} \mapsto W_d((W_u a_{t+i}^{(l)} + \mathbf{W_s g_t}) \odot \sigma(W_g a_{t+i}^{(l)})). \tag{2}$$

This ensures that the added overhead is negligible compared to the latency of the transformer, while allowing for a large amount of control in all layers of the drafter, as now the MLP is not solely conditioned on the hidden state, but also the steering vector. As $W_s g_t$ is invariant to drafting position i, we can compute it once at the beginning of each drafting stage. Note that the steering vector also influences the keys/ values in the attention mechanism, meaning the computation of $\pi_D(\hat{x}_{t+i} \mid x_{1:t}, g_t, \hat{x}_{t+1:t+i-1})$ is not only conditioned on g_t, but also all prior steering vectors $g_{t'}$ used in prior tokens.

Table 1: Block efficiency and speedup across a variety of tasks for $k = 8$ and $T = 1$ (See App. A.7 for $T = 0$), where UltraChat serves as the held-out validation set of training data for SD^2 and distilled. We report the block efficiency τ ($\pm$ denotes standard deviation over three independent evaluation runs; we further discuss statistical significance in App. A.5) and speedup α over pretrained drafters for each verifier/drafter combination, ordered by decreasing drafter-to-verifier size ratio. Particularly for smaller pretrained drafters, as in the example of Vicuna 1.3, incorporating steering mechanisms significantly enhances throughput, achieving on average 61% greater throughput and a 57% increase in block efficiency compared to its pretrained counterpart under standard sampling. Llama 3.1's pretrained drafter already demonstrates higher block efficiency overall (0.94 higher block efficiency than Qwen3 8B & Qwen3 0.6B on average), suggesting a naturally strong alignment between drafter and verifier. While distillation generally degrades performance across tasks not seen during training, SD^2 consistently preserves it. For Qwen and Llama models, both distillation and SD^2 fail to improve over pretrained drafters that are already well-aligned on GSM8K and HumanEval datasets. Notably, SD^2 always achieves higher block efficiency than its distilled counterpart and consistently also achieves greater throughput.

Method	**UltraChat**		**HumanEval**		**XSum**		**Alpaca**		**GSM8K**		**Mean**	
	τ	α	τ	α	τ	α	τ	α	τ	α	τ	α
Vicuna 1.3 13B & Llama 160M												
Pretrained	$1.93^{\pm 0.02}$	1.00	$1.68^{\pm 0.02}$	1.00	$2.08^{\pm 0.03}$	1.00	$1.83^{\pm 0.02}$	1.00	$1.90^{\pm 0.02}$	1.00	$1.88^{\pm 0.02}$	1.00
Distilled	$2.90^{\pm 0.04}$	1.53	$2.50^{\pm 0.00}$	1.53	$2.13^{\pm 0.04}$	0.97	$2.50^{\pm 0.03}$	1.39	$2.22^{\pm 0.01}$	1.19	$2.45^{\pm 0.02}$	1.32
SD^2	$\mathbf{3.45^{\pm 0.06}}$	**1.83**	$\mathbf{3.19^{\pm 0.08}}$	**1.96**	$\mathbf{2.46^{\pm 0.02}}$	**1.14**	$\mathbf{2.99^{\pm 0.03}}$	**1.67**	$\mathbf{2.72^{\pm 0.03}}$	**1.46**	$\mathbf{2.96^{\pm 0.04}}$	**1.61**
Qwen3 14B & Qwen3 0.6B												
Pretrained	$3.09^{\pm 0.03}$	1.00	$4.89^{\pm 0.07}$	1.00	$3.14^{\pm 0.04}$	1.00	$2.86^{\pm 0.04}$	1.00	$5.33^{\pm 0.08}$	1.00	$3.86^{\pm 0.05}$	1.00
Distilled	$3.59^{\pm 0.05}$	1.20	$4.88^{\pm 0.06}$	1.01	$3.09^{\pm 0.07}$	1.02	$3.14^{\pm 0.05}$	1.12	$5.16^{\pm 0.09}$	0.98	$3.97^{\pm 0.06}$	1.05
SD^2	$\mathbf{3.87^{\pm 0.02}}$	**1.28**	$\mathbf{5.25^{\pm 0.14}}$	**1.08**	$\mathbf{3.39^{\pm 0.03}}$	**1.10**	$\mathbf{3.39^{\pm 0.05}}$	**1.19**	$\mathbf{5.40^{\pm 0.07}}$	**1.01**	$\mathbf{4.26^{\pm 0.06}}$	**1.11**
Qwen3 8B & Qwen3 0.6B												
Pretrained	$3.17^{\pm 0.07}$	1.00	$\mathbf{5.18^{\pm 0.09}}$	**1.00**	$3.19^{\pm 0.03}$	1.00	$3.02^{\pm 0.04}$	1.00	$5.30^{\pm 0.01}$	**1.00**	$3.97^{\pm 0.05}$	1.00
Distilled	$3.71^{\pm 0.04}$	1.18	$5.10^{\pm 0.14}$	0.99	$3.16^{\pm 0.02}$	0.98	$3.20^{\pm 0.03}$	1.06	$5.16^{\pm 0.06}$	0.98	$4.07^{\pm 0.06}$	1.03
SD^2	$\mathbf{3.96^{\pm 0.05}}$	**1.24**	$5.18^{\pm 0.07}$	0.99	$\mathbf{3.40^{\pm 0.02}}$	**1.05**	$\mathbf{3.54^{\pm 0.07}}$	**1.16**	$\mathbf{5.31^{\pm 0.11}}$	0.99	$\mathbf{4.28^{\pm 0.06}}$	**1.06**
Llama 3.1 8B & Llama 3.2 1B												
Pretrained	$4.44^{\pm 0.03}$	1.00	$6.43^{\pm 0.07}$	**1.00**	$3.96^{\pm 0.05}$	1.00	$4.11^{\pm 0.16}$	1.00	$\mathbf{5.62^{\pm 0.08}}$	**1.00**	$4.91^{\pm 0.08}$	1.00
Distilled	$4.58^{\pm 0.03}$	1.03	$6.25^{\pm 0.07}$	0.97	$3.76^{\pm 0.02}$	0.94	$4.07^{\pm 0.02}$	0.99	$5.22^{\pm 0.05}$	0.93	$4.78^{\pm 0.04}$	0.97
SD^2	$\mathbf{4.79^{\pm 0.09}}$	**1.07**	$\mathbf{6.49^{\pm 0.11}}$	0.99	$\mathbf{4.07^{\pm 0.05}}$	**1.02**	$\mathbf{4.22^{\pm 0.08}}$	**1.02**	$5.44^{\pm 0.06}$	0.95	$\mathbf{5.00^{\pm 0.08}}$	**1.00**

3.3 Training

To train SD^2 we utilize synthetic data generated by π_V and use the probability distribution $\pi_V(x_t|x_{1:t-1})$ as targets. Similar to Zhou et al. [50], we use a synthetic dataset, as they have shown better alignment improvements compared to ground truth data, as it better reflects the verifier's behavior at inference. To train the steering mechanism, we utilize a uniformly random offset $\delta \in [1, k]$ and compute $\pi_D(x_t|x_{1:t-1}, g_{t-\delta})$. This ensures that the steering mechanism uniformly receives gradients for all drafting positions and hence must learn to encode information about the upcoming k tokens. In addition to the steering mechanism, we also fully fine-tune the drafter, while the verifier remains frozen throughout training to ensure lossless acceleration. This step is critical to the performance improvement of SD^2, as observed in our ablation presented in Appendix A.4. Leviathan et al. [21] showed that total variational distance (D_{TVD}) is equivalent to the rejection rate, making it the natural choice as a criterion. However, Zhou et al. [50] showed that the choice of loss is more nuanced and showed that Kullback–Leibler divergence (D_{KL}), which we adopt, often outperforms D_{TVD} as a criterion. The initialization of the steering mechanism is crucial, as too much interference by the untrained mechanism can lead to the model diverging. We initialize $W_s = 0$ and W_{hml} such that $W_{hml}[h_t, m_t, l_t]^\top = h_t + m_t + l_t$.

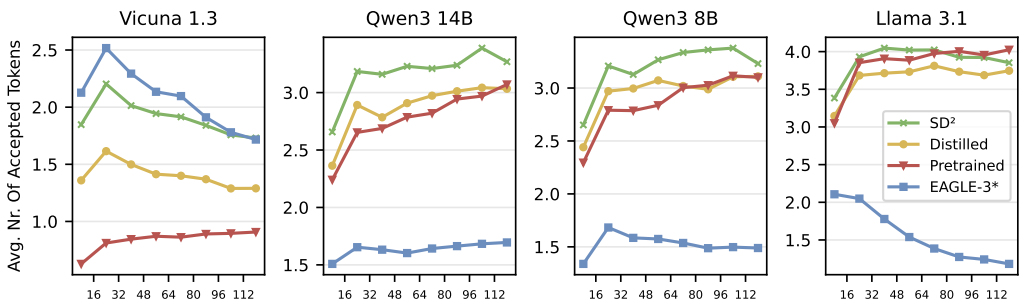

Figure 4: Number of tokens accepted per block at different positions. We compare how different drafter/verifier pairs fare at different positions throughout the generation process: A point at position x means the average number of accepted tokens per block for blocks with the last generated token having position $x \pm 8$. As can be seen, large pretrained drafters can leverage their vast training data to maintain strong drafting performance with increased sequence length. SD^2 minimally interferes with this behavior.

4 Experiments

Baselines. We evaluate the efficacy of our method against several drafting strategies: *Pretrained*, which employs speculative decoding with the unchanged drafter and *Distilled* [50], which first aligns the drafter to the verifier at training time. Additionally, we evaluate EAGLE-3* [24], a state-of-the-art dependent drafter used as a baseline for block efficiency. The asterisk indicates that we restrict EAGLE-3 to chain drafting mode to ensure a fair comparison and consistency with the other models. For SD^2 we set the l, m, h layers to 3, $\frac{L}{2}$, $L-2$ inspired by EAGLE-3.

Model Configurations. To assess the performance of SD^2, we use 4 different open source verifier-drafter pairs: *Vicuna 1.3* 13B with Llama 160M, *Qwen3 14B* and Qwen3 0.6B, *Qwen3 8B* and Qwen3 0.6B, and *Llama 3.1* 8B-Instruct and Llama 3.2 1B-Instruct. [49, 30, 46, 28, 29] These configurations were selected to represent a range of verifier–drafter capacity gaps and model families. For EAGLE-3*, we use the publicly released weights trained on UltraChat and ShareGPT datasets [24].

Tasks. We run experiments on 96 samples from 5 different datasets: The held-out validation split of *UltraChat_200k* [10] for dialogue, *HumanEval* [7] for code generation, *XSum* [31] for summarization, *Alpaca* [12] for instruction-following, and *GSM8K* [8] for reasoning. These common datasets provide coverage across core capabilities such as reasoning, summarization, and interaction. From this list, UltraChat_200k is the only dataset that the distilled and SD^2 drafters have seen during training.

Decoding Parameters. We fix the drafter's draft length to $k = 8$ tokens, yielding speculation blocks of size $k + 1 = 9$. All decoding is performed using the chain drafting strategy. We consider two sampling regimes: full sampling with temperature $T = 1$, and greedy decoding with $T = 0$. We use batched speculative decoding with a batch size of 12 and generate up to 128 output tokens per example. To ensure statistical reliability under stochastic sampling, we generate outputs at $T = 1$ across three distinct random seeds and report their mean and standard deviation. Due to memory limitations, we limit the total number of tokens computed (including rejected ones) to 512 tokens. For Vicuna 1.3, we relax this constraint by reducing the batch size and disabling the maximum token count, due to the low acceptance rate of the pretrained model.

Metrics. Since speculative decoding preserves the base model's probability distribution, this study focuses solely on efficiency metrics: *Block efficiency (τ)* and *speedup compared to the pretrained independent drafter (α)*. Block efficiency refers to the tokens generated per block, and is a driving factor in the efficacy of speculative decoding. This metric can be derived from the number of accepted tokens per block, and adding 1 for the token generated from the joint drafter-verifier distribution after rejection. We measure speedup as the increase in tokens generated per second compared to the

independent pretrained drafter. Note that speedup is hardware-dependent, unlike hardware-agnostic metrics such as τ.

Training Details. The distilled and SD^2 drafters are initialized from the pretrained drafter and finetuned for 6 epochs on synthetic data generated by the verifier with temperature $T = 1$, using prompts sourced from UltraChat_200k [10], limited to a total sequence length of 256. Training is conducted with an effective batch size of 24 using the AdamW [27] optimizer. Refer to Appendix A.3 for more info. After training on UltraChat, we fine-tune each drafter for one additional epoch on synthetic samples derived from the ShareGPT dataset [36]. Experiments are all performed on one NVIDIA A100 GPU with 80GB of memory. We release our code in Appendix A.2.

4.1 Results

Block Efficiency Table 1, Appendix A.7 and Fig. 5 summarize the results across all configurations. SD^2 consistently yields a higher block efficiency compared to both distilled and pretrained approaches. This improvement is particularly pronounced on UltraChat, the evaluation set from the training distribution, where SD^2 shows clear advantages. Across all datasets, SD^2 either matches or surpasses the performance of the pretrained model. As can be seen in Fig. 5, the Qwen and especially the Llama pretrained drafters already achieve high acceptance rates. This is particularly true in GSM8K and HumanEval, as can be seen in Table 1, where the distilled drafter is consistently outmatched by the pretrained drafter. This suggests that the distilled version has overfit to tasks in the style of UltraChat dialogue. While SD^2 also degrades in performance on these tasks compared to UltraChat or similar tasks, it can consistently match or beat both pretrained and distilled drafters. In the case of Vicuna 1.3, the pretrained drafter is not closely aligned to the model. This is to be expected, as in comparison to the other drafter-verifier pairs, these models have been trained on different data distributions and have a significantly larger capacity gap. In that setting, as observed in Table 1, both distillation and SD^2 significantly increase the block efficiency. For instance, with T=1 sampling, the distilled drafter achieves an average block efficiency improvement of 0.57 over the pretrained baseline across all tasks, while SD^2 further adds 0.51 accepted tokens per block. As seen in Fig. 5, both distilled and SD^2 perform reliably under both $T = 0$ (greedy decoding) and $T = 1$ (sampling), demonstrating robustness to different decoding regimes. Overall, the integration of steering leads to an average increase of 21% on the number of tokens accepted per block ($\tau - 1$) compared to distilled drafters and 31% compared to pretrained drafters

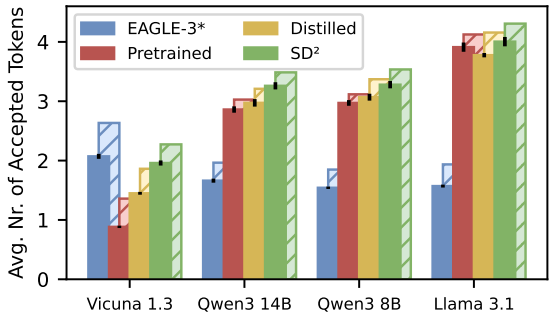

Figure 5: The average number of accepted tokens per Block for the different speculative decoding setups (Left to right: EAGLE-3*, Pretrained, Distilled, SD^2) averaged across all tasks. Solid bars correspond to $T = 1$ (sampling), and the hashed bars to $T = 0$ (greedy). One can see that SD^2 consistently achieves higher acceptance rates compared to both the Distilled and Pretrained drafter. In Vicuna 1.3, the number of active parameters for the drafter (Llama 160M) is less than half as many as the respective EAGLE-3* model. At such small sizes, pretrained drafters lose their competitiveness to dependent heads; however, SD^2 can bridge this gap.

Performance in Long Sequence Drafting. A key advantage of using pretrained drafters is their exposure to large-scale datasets and long-context training, which equips them with strong generation capabilities over extended sequences. As demonstrated in Fig. 4, both distilled and SD^2 maintain this capability. SD^2 consistently has more accepted tokens, and therefore also higher block efficiency, compared to both the pretrained and the distilled drafter across a range of token positions. Moreover, as evident by Fig. 4, SD^2 maintains a relatively constant advantage over distillation across all token positions, showing that steering works well with increasing sequence length. A continuation of the example in Table 2 is available in Appendix A.6.

Table 2: Qualitative example of speculative decoding with different drafting methods. *Green tokens* are accepted, ~~*red tokens*~~ are rejected, and the *blue token* is the final token per block sampled from the joint drafter-verifier distribution. Note that the symbol [?] refers to tokens outside of the English alphabet, highlighting the inherent risk of hidden state intervention. Continuation and more examples can be found in Appendix A.6

Pretrained	**Distilled**	**SD^2 (ours)**
~~Here is the implemented Python function 'has~~ To solve this problem, we need to ~~**~~ determine if ~~there exists at least two elements in~~ ~~**~~ any ~~pair of numbers in a list is~~ two numbers ~~in a list** are **clo~~ ~~**~~ ...	~~Here's the implemented Python function 'has~~ To solve this problem, you need to ~~:\n\n~~ determine whether ~~there exists any two numbers in the~~ ~~**~~ any ~~pair of numbers in a list is~~ two numbers ~~in the list** differ~~...	To solve the problem~~, we need to~~ of determining whether any two numbers in a list are closer to each other than a given 'threshold', we can ~~use a **hash map~~ approach it ~~as follows:\n\n### [?][?] Approach~~...

Speedup over Pretrained. Table 1 shows that, despite adding a small amount of computational latency to the drafting operation, SD^2 can speed up pretrained models by up to +83% on training data, while distilled models achieve an improvement of up to +53% over baseline. As evidenced by Tables 1 and Appendix A.7, across all tasks in Vicuna 1.3, SD^2 achieves a speedup of +61% under regular sampling and +43% under greedy sampling. On average, SD^2 provides a speedup of +19.5% for $T = 0$ (See Appendix A.7) and +16.3% for $T = 1$ compared to its pretrained counterpart. Crucially, SD^2 achieves roughly twice the additive speedup compared to distilled drafting under standard sampling and roughly 75% more additive speedup with greedy sampling. Furthermore, for Qwen3 8B on HumanEval with T=1, we observe that the steered method incurs only a 1% slowdown while matching the block efficiency of independent drafting, confirming the mechanism's minimal overhead.

5 Limitations and Future Work

The performance of SD^2, much like distillation-based approaches, is highly dependent on the composition and quality of the training data. Although SD^2 often matches the pretrained drafter on out-of-domain tasks, its effectiveness remains strongest on data similar to its training distribution, as shown in Table 1. This highlights the importance of either training on a comprehensive and diverse dataset or limiting the drafter to a singular domain. Furthermore, while achieving higher block efficiency, it provides little to no speedup over an already well-aligned drafter, such as Llama 3.1. Moreover, changing hidden representations in transformer networks is a delicate matter, as small changes in the wrong direction, as evidenced in Table 2, can lead the model to produce nonsensical output. Additionally, while speculative decoding with pretrained drafters can isolate the verifier in a black box, SD^2 requires access to the verifier's hidden states. This can be challenging in applications involving external remote verifiers. While we demonstrate that steering can be retrofitted onto existing drafters, we do not explore the training of new drafters explicitly designed for dynamic steering, which we leave as an open direction for future work. Furthermore, we do not compare SD^2 's steering to more invasive methods like EAGLE's concatenation of verifier states. However, SD^2 's key advantage is its modularity, as steering can be added post hoc without requiring verifier signals during pretraining. All models in this study use SwiGLU [37] in their feedforward layers. While our steering mechanism should generalize to other gated activations, this remains to be validated in future work. Finally, extending SD^2 to more complex speculative decoding paradigms, such as dynamic tree verification, remains an open problem, with application-specific studies needed to assess its practical viability and competitiveness against other speculative decoding paradigms.

6 Conclusion

This study presents a method to dynamically steer pretrained drafters during speculative decoding, achieving substantial performance improvements compared to baselines and across a wide range of drafter and verifier configurations. In addition to improving acceptance rates, our system exhibits greater robustness on out-of-distribution tasks, suggesting that steering mechanisms are less susceptible to over-fitting on the training task.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 Technical Report, 2023.

[2] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-Dependent Draft Heads for Medusa Decoding. In *Proceedings of the Conference on Language Modeling (CoLM)*, 2024.

[3] F. Warren Burton. Speculative Computation, Parallelism, and Functional Programming. *IEEE Transactions on Computers*, 100(12):1190–1193, 2012.

[4] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[5] Sviatoslav Chalnev, Matthew Siu, and Arthur Conmy. Improving Steering Vectors by Targeting Sparse Autoencoder Features, 2024.

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

[7] 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. Evaluating Large Language Models Trained on Code, 2021.

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training Verifiers to Solve Math Word Problems, 2021.

[9] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[10] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing Chat Language Models by Scaling High-Quality Instructional Conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[11] Cunxiao Du, Jing Jiang, Yuanchen Xu, Jiawei Wu, Sicheng Yu, Yongqi Li, Shenggui Li, Kai Xu, Liqiang Nie, Zhaopeng Tu, et al. GliDe with a CaPE: A Low-Hassle Method to Accelerate Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[12] Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-Controlled AlpacaEval: A Simple Way to Debias Automatic Evaluators, 2024.

[13] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[14] Tao Ge, Heming Xia, Xin Sun, Si-Qing Chen, and Furu Wei. Lossless Acceleration for Seq2seq Generation with Aggressive Decoding, 2022.

[15] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 Herd of Models, 2024.

[16] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network, 2015.

[18] Yunhai Hu, Zining Liu, Zhenyuan Dong, Tianfan Peng, Bradley McDanel, and Sai Qian Zhang. Speculative Decoding and Beyond: An In-Depth Survey of Techniques, 2025.

[19] Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths, 2024.

[20] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse Autoencoders Find Highly Interpretable Features in Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[21] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

[22] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[23] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[24] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling Up Inference Acceleration of Large Language Models via Training-Time Test, 2025.

[25] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 Technical Report, 2024.

[26] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[27] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[28] Meta AI. Introducing Llama 3.1: Our Most Capable Models to Date. `https://ai.meta.com/blog/meta-llama-3-1/`, 2024. Accessed: 2025-07-29.

[29] Meta AI. Llama 3.2: Revolutionizing Edge AI and Vision with Open, Customizable Models. `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`, 2024. Accessed: 2025-07-29.

[30] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. SpecInfer: Accelerating Large Language Model Serving with Tree-Based Speculative Inference and Verification. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[31] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[32] OpenAI. Predicted Outputs Guide. `https://platform.openai.com/docs/guides/predicted-outputs`, 2024. Accessed: 2025-07-29.

[33] Kiho Park, Yo Joong Choe, and Victor Veitch. The Linear Representation Hypothesis and the Geometry of Large Language Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024.

[34] Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering Llama 2 via Contrastive Activation Addition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[35] Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your LLM Knows the Future: Uncovering Its Multi-Token Prediction Potential, 2025.

[36] ShareGPT. ShareGPT. `https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered`, 2023. Accessed: 2025-07-29.

[37] Noam Shazeer. GLU Variants Improve Transformer, 2020.

[38] Benjamin Frederick Spector and Christopher Re. Accelerating LLM Inference with Staged Speculative Decoding, 2023.

[39] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise Parallel Decoding for Deep Autoregressive Models. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[40] Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. Instantaneous Grammatical Error Correction with Shallow Aggressive Decoding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

[41] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. SpecTr: Fast Speculative Decoding via Optimal Transport. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[42] Ziteng Sun, Uri Mendlovic, Yaniv Leviathan, Asaf Aharoni, Ahmad Beirami, Jae Hun Ro, and Ananda Theertha Suresh. Block Verification Accelerates Speculative Decoding, 2024.

[43] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering Language Models with Activation Engineering, 2023.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[45] Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative Decoding: Exploiting Speculative Execution for Accelerating Seq2seq Generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[46] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report, 2025.

[47] Ming Yin, Minshuo Chen, Kaixuan Huang, and Mengdi Wang. A Theoretical Perspective for Speculative Decoding Algorithm. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[48] Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning Harmonized Representations for Speculative Sampling. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[49] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[50] Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[51] Matthieu Zimmer, Milan Gritta, Gerasimos Lampouras, Haitham Bou Ammar, and Jun Wang. Mixture of Attentions for Speculative Decoding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

A Technical Appendix

A.1 Notation

- π_V: Verifier model (also referred to as the base model).
- π_D: Drafter model.
- k: Number of tokens generated (drafted) per step.
- τ: Block efficiency metric.
- α: Speed up in throughput compared to using the pretrained drafter.
- $x_{i:j} := \{x_k \mid i \leq k \leq j\}$: Token subsequence from position i to j, inclusive.
- g_t: Steering vector generated by π_V at timestep t.
- h_t, m_t, l_t: High-, middle-, and low-level hidden states from π_V at timestep t.
- $f_t^{(l)}$: Hidden state at timestep t after the l-th layer.
- $T = 0, T = 1$: Sampling temperatures; $T = 1$ corresponds to standard sampling, and $T = 0$ to greedy sampling.
- $[a, b] := \{i \in \mathbb{N} \mid a \leq i \leq b\}$: Closed interval over natural numbers from a to b.

A.2 Code

We release our code to support full reproducibility and to facilitate further research.

`https://github.com/ETH-DISCO/SD-square`

A.3 Training Details

Training was done on 1xA100-80GB with BF-16 precision. The drafter was stored in full precision throughout training. We use a cosine learning rate scheduler with 1000 warm-up steps starting and ending at 1/10th the learning rate. We employ gradient-norm clipping with a value of 0.5 to ensure stability. For AdamW we set $\beta_1 = 0.9$, $\beta_2 = 0.999$, and model-specific learning rates (5e-5 for Vicuna 1.3, 1e-5 for Llama 3.1 and 2e-5 for Qwen3 8B & 14B).

A.4 Ablation Studies

We justify the design of the steering mechanism and training in SD^2 with ablation studies on *Vicuna-1.3 7B* and *Llama 160M*. We investigate three key SD^2 design choices: The steering mechanism, unfreezing the drafter, and the offsets used during training.

What steering mechanism is most effective? The choice of a steering mechanism, i.e, how we modify the behavior of the drafter conditioned on the steering vector g_t, is critical for the effectiveness of SD^2. We aim to design an optimal steering mechanism that (i) is latency-lightweight, (ii) remains compatible with other acceleration techniques like KV-Caching, and (iii) provides precise control over the drafter's behavior. We evaluated three options that differ in the number of parameters and expressiveness. The simplest method adds a bias $W_s g_t$ right after the MLP for all hidden layers, so $\tilde{f}_{t+i}^{(l)} := f_{t+i}^{(l)} + W_s g_t$ for all $i \in [1, k]$. Note that $W_s g_t$ only has to be computed once, as it is invariant of draft position i. The second approach, which we ultimately adopt in SD^2 (see Eq. 2), modifies this by instead conditioning the existing MLP on g_t for all layers by adding $W_s g_t$ to the up-projection, right before gating. The last approach modifies all layers by adding a 2-layer MLP with input $f_t^{(l-1)}$ and g_t, which computes the bias, which in turn is then used inside the MLP as in the second approach. As seen in Fig. 6, SD^2 consistently scores the highest block efficiency.

Should one fine-tune the drafter? In SD^2, we fine-tune both the steering mechanism and the drafter parameters. To isolate the effect of steering, we also evaluate a frozen-drafter variant where only the steering is trained. As inferrable from Fig. 6, steering alone can increase the number of tokens accepted by +100% over the unaligned pretrained's baseline of 1.0 , indicating that the verifier's hidden states convey valuable guidance and can meaningfully influence the drafter's output. However, as Fig. 6 shows, unfreezing the drafter consistently yields better results.

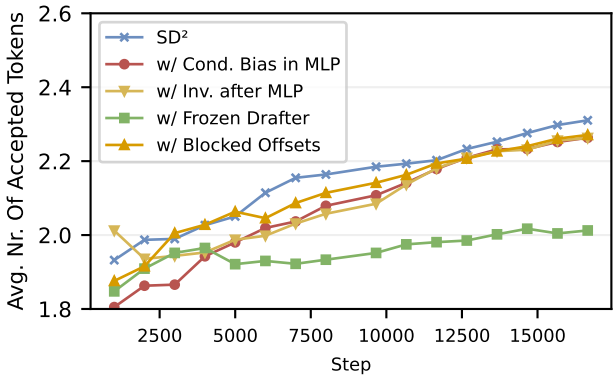

Figure 6: Number of tokens accepted for ablation experiments throughout training for 2 epochs on Vicuna 1.3 7B (π_V) and Llama 160M (π_D). We omit step 0, where every experiment has the value of the pretrained drafter, of 1.0. SD^2 utilizes a bias in the MLP, right after the up-projection, and unfreezes the drafter. *Inv. Bias after MLP* simplifies the steering mechanism of SD^2 by adding the bias right after the MLP, while *Cond. Bias in MLP* increases the modeling capability of the steering mechanism by instead calculating the bias based on not only g_t, but also $f_{t+i}^{(l-1)}$. We also test keeping the drafter frozen throughout training. Finally, in the *Blocked Offsets* experiment, we change the training mechanism to mimic offsets seen during inference by using g_t for the prediction of tokens $x_{t+1:t+k}$ (instead of using $g_{t-\delta}$ for the prediction of x_t with a random δ). SD^2 consistently outperforms the other variants, justifying our design choices.

Should one model the offsets used in training after inference? During training, our method samples a random offset $\delta \in [1, k]$ and uses the steering vector g_t to predict the distribution of $\hat{x}_{t+\delta}$. This differs from inference, where g_t conditions the prediction of all tokens $\hat{x}_{t+1:t+k}$. To better simulate inference behavior during training, we instead apply a *blocked offset* strategy: g_t is used as the steering vector for the entire prediction block $\hat{y}_{t+1:t+k}$, where $t \bmod k = \delta - 1$. As can be seen in Fig. 6, these two methods only differ slightly, with SD^2 having the slight edge.

A.5 Statistical Significance

We evaluate the statistical significance of our results using a pair-wise Welch's t-test across datasets and drafter-verifier pairs. Refer to Table 3

A.6 Example of speculative decoding

See Table 4 for a continuation of the example in Table 2.

A.7 Performance under Greedy Sampling

See Table 5 for the results under the greedy sampling paradigm.

Table 3: Pairwise t-tests on block efficiency across datasets and models for $T = 1$ across 3 distinct seeds. We evaluate the statistical significance of each of the results by considering a pair-wise t-test between the pretrained baseline and the distilled version or SD^2. t corresponds to Welch's t-statistic, which measures the standardized difference. A positive t-statistic indicates an improvement over the pretrained drafter. p corresponds to the probability of the null hypothesis, i.e., that there is no difference between the two distributions. We marked all $p \leq 0.05$ in bold. As is evident, both distilled and SD^2 make improvements over the pretrained drafter on the held-out validation data from the training dataset UltraChat_200k. Meanwhile, especially for Llama 3.1, the difference in block efficiency between SD^2 and pretrained drafters is less clear as the two models tend to perform similarly on out-of-training-data evaluations. For Vicuna, one can see that SD^2 performs better over the pretrained baseline with very high probability

Method	**UltraChat**		**HumanEval**		**XSum**		**Alpaca**		**GSM8K**	
	t	p	t	p	t	p	t	p	t	p
Vicuna 1.3 13B & Llama 160M										
Distilled	40.15	**0.00**	72.90	**0.00**	1.82	0.16	35.19	**0.00**	32.51	**0.00**
SD^2	43.70	**0.00**	32.96	**0.00**	18.65	**0.00**	54.96	**0.00**	47.49	**0.00**
Qwen3 14B Qwen3 0.6B										
Distilled	13.91	**0.00**	-0.26	0.81	-1.04	0.37	8.19	**0.00**	-2.41	0.07
SD^2	35.40	**0.00**	3.88	**0.03**	9.20	**0.00**	14.56	**0.00**	1.19	0.30
Qwen3 8B Qwen3 0.6B										
Distilled	11.43	**0.00**	-0.83	0.46	-1.54	0.20	6.12	**0.00**	-4.03	0.05
SD^2	15.99	**0.00**	-0.05	0.96	9.80	**0.00**	11.94	**0.00**	0.24	0.83
Llama 3.1 8B Llama 3.2 1B										
Distilled	6.35	**0.00**	-3.14	**0.03**	-6.28	**0.02**	-0.48	0.68	-7.60	**0.00**
SD^2	6.55	**0.01**	0.85	0.45	2.56	0.06	1.06	0.37	-3.25	**0.04**

Table 4: Continuation of qualitative example Table 2 of speculative decoding with different drafting methods using $T = 1$ sampling and the Qwen3 14B verifier/drafter setup. *Green tokens* are accepted, ~~*red tokens*~~ are rejected, and the *blue token* is the final token per block sampled from the joint drafter-verifier distribution.

Pretrained	**Distilled**	**SD^2 (ours)**
~~Here is the implemented Python function ‘has~~ To solve this problem, we need to ~~**~~ determine if ~~there exists at least two elements in~~ ~~**~~ any ~~pair of numbers in a list is~~ two numbers ~~in a list** are **clo~~ ~~**~~ in the ~~list ‘numbers‘ are **~~ given list ~~of floating-point numbers are **clo~~ are ~~closer to each other than a specified **~~ ~~**~~ closer to each other than the given threshold **.\n\n### ~~[?][?] Type hints (~~ Strategy :\n\n- ~~** Check all pairs** of~~ Sort the list** of numbers ~~.\n- **~~ first ~~.\n- **Check adjacent pairs** in~~ . ~~\n- **Iterate through the sorted~~ When ~~sorted, two numbers that are close to~~ the list is sorted, the closest ~~elements to~~ numbers ~~between two elements are at the **ends~~ will be ~~near the ends of the list.\n~~ ~~**~~ adjacent** ~~in the list, and~~ ~~.\n~~ - ~~Use a **linear scan** to~~ Then , ~~check if any pair of **adj~~ iterate through the sorted list and compare ~~each pair~~ ~~**~~ each ~~number with the previous one** to~~ pair of adjacent numbers** ~~with the threshold.\n~~ ~~.\n~~ - If ~~any such pair has elements that~~ the ~~difference between them is **less than the~~ ~~**~~ ~~difference between two consecutive elements is less than~~ absolute difference ~~between any two adjacent numbers** is~~ ~~**~~ between any two adjacent numbers ~~**less than~~ is **less than the threshold**, return ‘ True ‘.\n- If no such pair is found after	~~Here’s the implemented Python function ‘has~~ To solve this problem, you need to ~~:\n\n~~ determine whether ~~there exists any two numbers in the~~ ~~**~~ any ~~pair of numbers in a list is~~ two numbers ~~in the list** differ by **~~ ~~**~~ in the list are **closer ~~than~~ to each other than the given threshold**.~~ This ~~is a **common problem** found in~~ can be done ~~in a few key steps:\n\n~~ efficiently ~~using a **hash map (dictionary)**~~ by ~~checking all possible pairs of elements (i~~ ~~**~~ ~~finding the minimum distance** between any two~~ sorting the list**, and then checking ~~for elements~~ the ~~pairs later based on their element and its~~ ~~**~~ ~~distance between neighboring elements**, since saying that~~ difference between each pair of consecutive elements**.\n\n ### Key Steps:\n\n1. **Sort the list ** ~~so that the smallest number comes first – This helps in easily finding neighboring elements.\n~~ Sorting ~~the array allows us to easily iterate through~~ makes it easier to ~~analyze consecutive elements.\n2~~ check for adjacent elements, because the ~~list becomes~~ closest elements will be next to each other ~~.\n~~ in ~~the sorted list.\n\n2. **Iter~~ a sorted list.\n2. **Check adjacent elements ** – For each adjacent pair in the sorted list ~~(i.e., ‘sorted_numbers~~ , compute the absolute difference ~~between the two.\n~~ . If any ~~of these differences **less than~~ difference is less than the ~~given threshold, return~~ threshold , return ‘True‘.\n3. ** Return	To solve the problem ~~, we need to~~ of determining whether any two numbers in a list are closer to each other than a given ‘ threshold ‘, we can ~~use a **hash map~~ approach it ~~as follows:\n\n### [?][?] Approach~~ systematically .\n\nThe ~~idea is to:\n\n1.~~ key idea is ~~that we can:\n\n1.~~ :\n\n - ~~Compare **all** pairs of distinct~~ Sort the list of numbers ~~in ascending order.\n~~ , ~~which helps in efficiently comparing all pairs (~~ because after sorting, the ~~indices of elements that~~ closest ~~values to each other (i.e.,~~ pair ~~will always be the second and second-to~~ of numbers will be ~~consecutive elements in the sorted~~ adjacent ~~in the original list.\n- Traverse the~~ .\n - ~~For each element in the sorted list~~ Then , iterate through the sorted list and check if the difference between ~~the current number and the~~ any two adjacent numbers is less than the ~~given~~ ‘ threshold‘.\n\nHere’s the implementation ~~in Python~~ of the function ~~you provided, with a bit~~ with the correct ~~‘threshold‘, using ‘typing~~ logic :\n\n“‘python\nfrom typing import List \n\n def has_close_elements(numbers: List[float], threshold: float) ->bool:\n """ Check if in given list of numbers, any two numbers are closer to each other\n than ~~given threshold.\n \n »>ha~~ the given threshold.\n\n »>has_close_elements ([~~than given threshold.\n \n »>ha~~ float ~~’given threshold.\n »>has_clos~~ \n

Table 5: Block efficiency and speedup across a variety of tasks for $k = 8$ and $T = 0$. We report the block efficiency τ and speedup α over the pretrained drafters throughput for each verifier/drafter combination, ordered by decreasing drafter-to-verifier size ratio. Both distilled drafters and SD^2 respond well to $T = 0$ sampling and gain more benefit compared to $T = 1$ sampling (See Table 1). Under $T = 0$, SD^2 has 3.7% higher block efficiency then the Llama 3.2 pretrained drafter, compared to 1.8% under $T = 1$ sampling. Further more, SD^2 has the highest block efficiency for all tasks and models.

Method	**UltraChat**		**HumanEval**		**XSum**		**Alpaca**		**GSM8K**		**Mean**	
	τ	α	τ	α	τ	α	τ	α	τ	α	τ	α
Vicuna 1.3 13B & Llama 160M												
Pretrained	2.47	1.00	2.08	1.00	2.58	**1.00**	2.26	1.00	2.40	1.00	2.36	1.00
Distilled	3.35	1.39	3.01	1.48	2.45	0.92	2.86	1.30	2.64	1.12	2.86	1.24
SD^2	**3.83**	**1.59**	**3.63**	**1.80**	**2.62**	0.99	**3.26**	**1.48**	**3.03**	**1.28**	**3.27**	**1.43**
Qwen3 14B & Qwen3 0.6B												
Pretrained	3.13	1.00	5.17	1.00	3.30	1.00	2.98	1.00	5.57	1.00	4.03	1.00
Distilled	3.82	1.26	5.12	1.00	3.30	1.03	3.35	1.14	5.45	0.98	4.21	1.06
SD^2	**4.05**	**1.33**	**5.47**	**1.05**	**3.61**	**1.11**	**3.64**	**1.23**	**5.66**	**1.01**	**4.49**	**1.12**
Qwen3 8B & Qwen3 0.6B												
Pretrained	3.24	1.00	5.35	1.00	3.38	1.00	3.20	1.00	5.41	1.00	4.12	1.00
Distilled	3.93	1.23	5.44	**1.02**	3.46	1.03	3.55	1.12	5.47	1.01	4.37	1.07
SD^2	**4.15**	**1.28**	**5.51**	1.02	**3.71**	**1.09**	**3.73**	**1.16**	**5.58**	**1.02**	**4.54**	**1.09**
Llama 3.1 8B & Llama 3.2 1B												
Pretrained	4.51	1.00	6.73	**1.00**	4.11	1.00	4.41	1.00	5.86	**1.00**	5.12	1.00
Distilled	4.89	1.10	6.68	1.00	4.06	1.00	4.33	0.98	5.82	0.99	5.16	1.01
SD^2	**5.04**	**1.11**	**6.78**	0.99	**4.24**	**1.03**	**4.55**	**1.01**	**5.93**	0.99	**5.31**	**1.02**