

Steering Pretrained Drafters during Speculative Decoding

Frédéric Berdoz, Peer Rheinboldt, Roger Wattenhofer

ETH Zurich

{fberdoz, prheinboldt, 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.

Code — <https://github.com/ETH-DISCO/SD-square>

1 Introduction

The auto-regressive nature of transformer-based large language models (LLMs) (Vaswani et al. 2017) inherently limits their inference speed. This limitation is further amplified by the rapid growth in model size among frontier LLMs (Achiam et al. 2023; Grattafiori et al. 2024; Liu et al. 2024a; Yang et al. 2025). Numerous approaches have been proposed to reduce latency, including weight quantization (Dettmers et al. 2022), model pruning (Han, Mao, and Dally 2016), and distillation (Hinton, Vinyals, and Dean 2015), but these often come at the expense of generated text quality. A paradigm that escapes this trade-off is *speculative decoding* (Leviathan, Kalman, and Matias 2023; Xia et al. 2023; Chen et al. 2023), which follows the general principle of *speculative execution* (Burton 2012). 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 (Hu et al. 2025). The first uses an independent drafter (Xia et al. 2023), 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 (Stern, Shazeer, and Uszkoreit 2018; Cai et al. 2024; Ankner et al. 2024; Li et al. 2024b). 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 (Miao et al. 2024), this makes them competitive in settings where the cost of verification is relatively low, such as in controlled research environments. However, in real-world scenarios where verification latency fluctuates or dominates total runtime, e.g., when the verifier is remote (OpenAI 2024), deployed on slower hardware, shared across several drafters, or simply frontier-scale with 600B+ parameters (Liu et al. 2024a), 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 (Samragh et al. 2025), we propose **Steering pretrained Drafters during Speculative Decoding (SD²)**, 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 num-

This is the extended version of the corresponding paper published in the AAAI 2026 proceedings.

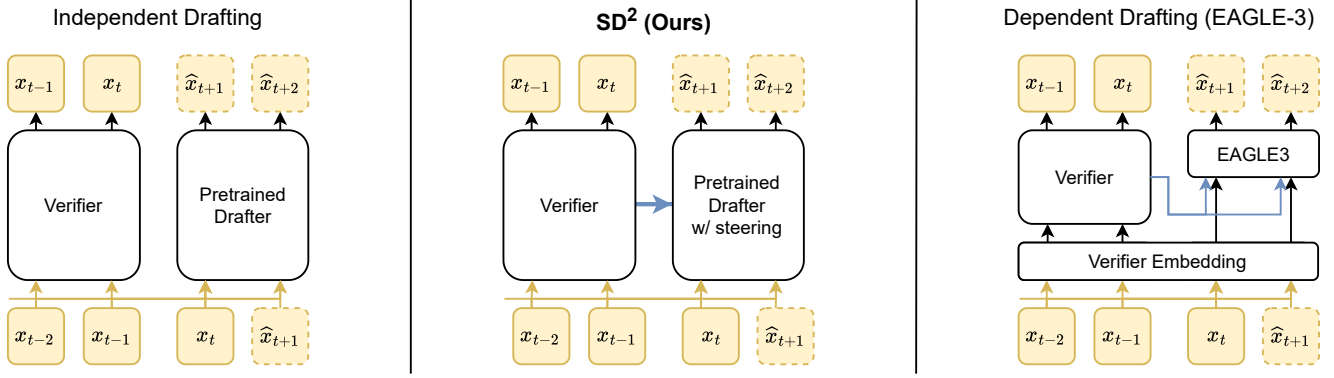


Figure 1: Overview of different drafting paradigms: Independent drafting uses a smaller model from the same family as the verifier, with no access to its internal state. Dependent drafting (e.g., EAGLE-3) uses lightweight heads trained to read the verifier’s hidden states, sharing input embeddings and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

ber 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 (Burton 2012). While early variants only supported greedy decoding acceleration (Stern, Shazeer, and Uszkoreit 2018; Sun et al. 2021; Ge et al. 2022; Xia et al. 2023), the concurrent works of Leviathan, Kalman, and Matias (2023) and Chen et al. (2023) 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. (2025) 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 (Stern, Shazeer, and Uszkoreit 2018; Cai et al. 2024). 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. (2024b) propose to use autoregressive drafters on the hidden states, and Ankner et al. (2024) 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. (2025) and Du et al. (2024) use the KV values of the verifiers during drafting. Zhang et al. (2025) and Li et al. (2025) further improve the acceptance rates by training the drafter to use its hidden features to close the gap between training and inference. Although our study centers

on independent drafters, we also report the block efficiency of EAGLE-3 (Li et al. 2025) 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. (2023). Building on this, Huang, Guo, and Wang (2024) proposed an enhanced version where the candidate length is determined on the fly via an acceptance prediction head. Zhou et al. (2024) 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, (Liu et al. 2024b) propose online speculative decoding, where drafters are continuously retrained on new user inputs, and Fu et al. (2024) propose a drafter-free version using intermediate Jacobi iterations as drafted sequences.

Verification. Sun et al. (2023, 2024) frame the verification phase as an optimal transport problem to improve batch and block verification, respectively. Spector and Re (2023) and Miao et al. (2024) 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. (2024a) introduce dynamic drafting trees. Lastly, (Yin et al. 2024) explore the theoretical limits of speculative decoding.

2.2 Dynamic Steering of LLMs

The technique of activation steering, first proposed by Turner et al. (2023), allows for the control of LLM behavior by directly modifying model activations during inference. It is primarily motivated by the *linear representation hypothesis* (Park, Choe, and Veitch 2024), suggesting that a model’s intent or behavior is encoded along specific, steerable directions. Subsequently, Rimsky et al. (2024) introduced a method to compute steering vectors by averaging the activation differences between sets of positive and negative examples. More recently, Chalnev, Siu, and Conmy

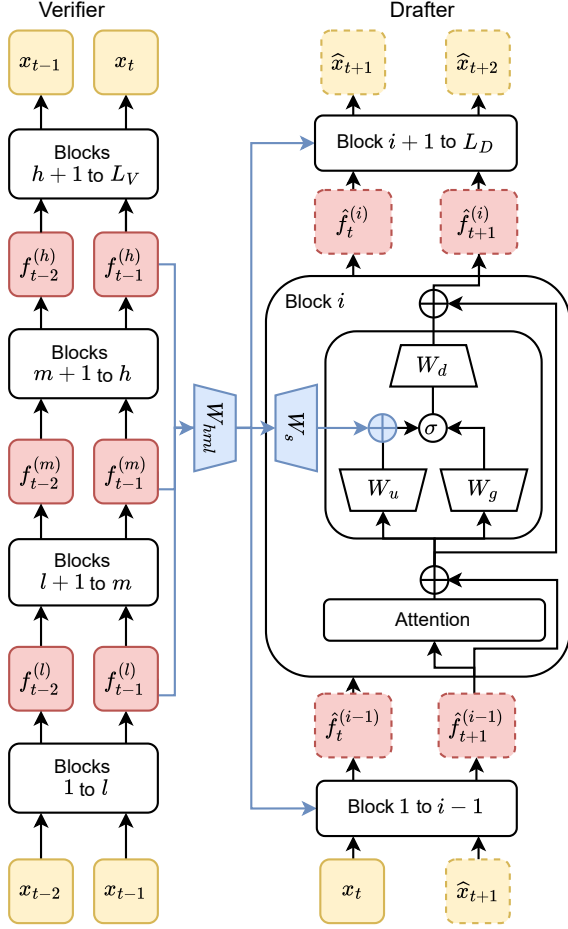


Figure 2: The steering mechanism in SD^2 works by concatenating the verifier’s high-, medium-, and low-level hidden features and passing them through a linear projection to produce a steering vector. This embedding is transformed by another linear layer into a set of biases, which are added to all MLP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

(2024) introduce a method to predict a steering vector’s impact on internal sparse autoencoder (SAE) features (Huben et al. 2024). However, these approaches focus on static, interpretable 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 (Leviathan, Kalman, and Matias 2023; Chen et al. 2023). The key addition is that, in addition 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 ex-

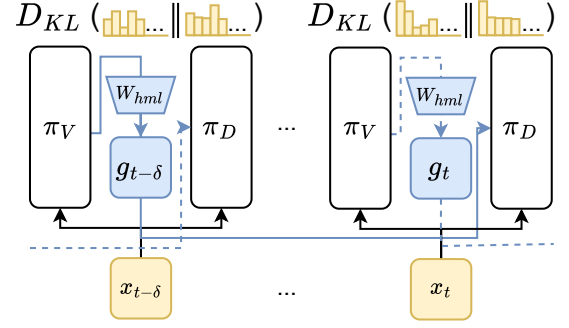


Figure 3: The training process of SD^2 aligns the drafter’s (π_D) probability distribution to the verifier’s (π_V). To achieve this, we randomly choose an offset $\delta \in [1, k]$ to simulate drafting the δ ’th token of a block. After extracting g from on the verifier’s activations, we compute $\pi_D(x_t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

PLICITLY trained to do so (Samragh et al. 2025). We aim to extract this predictive information from the verifier’s hidden representations and inject it into the drafter to dynamically guide generation.

3.1 Verification

In SD^2 , the verification of candidate tokens remains unchanged to the regular speculative decoding framework, where we compute $\pi_V(\hat{x}_{t+i} | x_{1:t}, \hat{x}_{t+1:t+i-1})$ 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 (Leviathan, Kalman, and Matias 2023; Yin et al. 2024). 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 (Li et al. 2025) 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 $g_t = W_{hml}[h_t, m_t, l_t]^T$.

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 $\pi_D(\hat{x}_{t+i} | x_{1:t}, g_t, \hat{x}_{t+1:t+i-1})$ 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 $l = 1, \dots, L_D$ of π_D by changing the SwiGLU (Shazeer 2020) computation from

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

Verifier & Drafter	Method	UltraChat		HumanEval		XSum		Alpaca		GSM8K		Mean	
		τ	α	τ	α	τ	α	τ	α	τ	α	τ	α
T=1 (Sampling)													
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	3.45$\pm$0.06	1.83	3.19$\pm$0.08	1.96	2.46$\pm$0.02	1.14	2.99$\pm$0.03	1.67	2.72$\pm$0.03	1.46	2.96$\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	3.87$\pm$0.02	1.28	5.25$\pm$0.14	1.08	3.39$\pm$0.03	1.10	3.39$\pm$0.05	1.19	5.40$\pm$0.07	1.01	4.26$\pm$0.06	1.11
Qwen3 8B Qwen3 0.6B	Pretrained	3.17 $\pm$ 0.07	1.00	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	3.96$\pm$0.05	1.24	5.18 $\pm$ 0.07	0.99	3.40$\pm$0.02	1.05	3.54$\pm$0.07	1.16	5.31$\pm$0.11	0.99	4.28$\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	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	4.79$\pm$0.09	1.07	6.49$\pm$0.11	0.99	4.07$\pm$0.05	1.02	4.22$\pm$0.08	1.02	5.44 $\pm$ 0.06	0.95	5.00$\pm$0.08	1.00
T=0 (Greedy)													
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

Table 1: Block efficiency and speedup across a variety of tasks for $k = 8$, where UltraChat serves as the held-out validation set of training data for SD² and distilled. We report the block efficiency τ ($\pm$ denotes standard deviation over three independent evaluation runs; we further discuss statistical significance in Section A.4) 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, SD² consistently preserves it. For Qwen and Llama models, both distillation and SD² fail to improve over pretrained drafters already well-aligned on GSM8K and HumanEval datasets. Notably, SD² always achieves higher block efficiency than distillation and consistently achieves greater throughput.

to

$$a_{t+i}^{(l)}, \mathbf{g}_t \mapsto W_d((W_u a_{t+i}^{(l)} + \mathbf{W}_s \mathbf{g}_t) \odot \sigma(W_g a_{t+i}^{(l)})). \quad (2)$$

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.

3.3 Training

To train SD² we utilize synthetic data generated by π_V and use the probability distribution $\pi_V(x_t \mid x_{1:t-1})$ as targets. Similar to Zhou et al. (2024), 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 $\pi_D(x_t \mid x_{1:t-1}, g_{t-\delta})$. 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

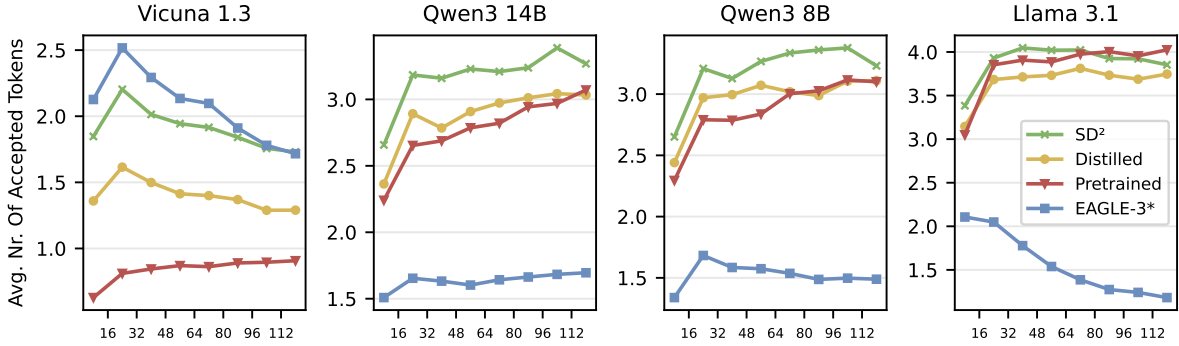


Figure 4: Number of tokens accepted per block at different positions. We compare how different drafter/verifier pairs fare at different positions throughout the generation process: A point at position x means the average number of accepted tokens per block for blocks with the last generated token having 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.

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 Fig. 6. Leviathan, Kalman, and Matias (2023) 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. (2024) 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 $W_{hml}[h_t, m_t, l_t]^\top = h_t + m_t + l_t$.

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* (Zhou et al. 2024), which first aligns the drafter to the verifier at training time. Additionally, we evaluate EAGLE-3* (Li et al. 2025), 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.

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. (Zheng et al. 2023; Miao et al. 2024; Yang et al. 2025; Meta AI 2024a,b) 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 (Li et al. 2025).

Tasks. We run experiments on 96 samples from 5 different datasets: The held-out validation split of *UltraChat.200k* (Ding et al. 2023) for dialogue, *HumanEval* (Chen et al.

2021) for code generation, *XSum* (Narayan, Cohen, and Lapata 2018) for summarization, *Alpaca* (Dubois et al. 2024) for instruction-following, and *GSM8K* (Cobbe et al. 2021) 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

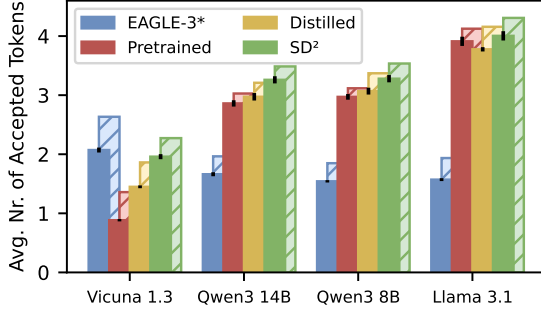


Figure 5: The average number of accepted tokens per Block for the different speculative decoding setups (Left to right: EAGLE-3*, Pretrained, Distilled, SD²) averaged across all tasks. Solid bars correspond to $T = 1$ (sampling), and the hashed bars to $T = 0$ (greedy). One can see that SD² consistently 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, SD² can bridge this gap.

epochs on synthetic data generated by the verifier with temperature $T = 1$, using prompts sourced from UltraChat_200k (Ding et al. 2023), limited to a total sequence length of 256. Training is conducted with an effective batch size of 24 using the AdamW (Loshchilov and Hutter 2019) optimizer. Refer to Section A.2 for more info. After training on UltraChat, we fine-tune each drafter for one more epoch on synthetic samples derived from the ShareGPT dataset (ShareGPT 2023). All experiments are performed on one NVIDIA A100 GPU with 80GB of memory.

4.1 Ablations

We justify the design of the steering mechanism and training in SD² with ablation studies on *Vicuna-1.3 7B* and *Llama 160M*. We investigate two key SD² design choices: The steering mechanism and unfreezing the drafter. Further ablation studies can be found in Section A.3.

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 SD². 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 SD² (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 mod-

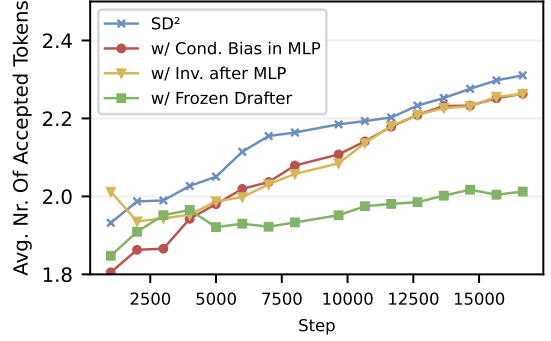


Figure 6: Number of tokens accepted for ablation experiments throughout training for 2 epochs on Vicuna 1.3 7B (π_V) and Llama 160M (π_D). We omit step 0, where every experiment has the value of the pretrained drafter, of 1.0. SD² utilizes a bias in the MLP, right after the up-projection, and unfreezes the drafter. *Inv. Bias after MLP* simplifies the steering mechanism of SD² 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 during training. SD² consistently outperforms the other variants, justifying our design choices.

ifies 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, SD² consistently scores the highest block efficiency.

Should one fine-tune the drafter? In SD², 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 inferable 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 shown in Fig. 6, unfreezing the drafter consistently yields better results.

4.2 Results

Block Efficiency Table 1 and Fig. 5 summarize the results across all configurations. SD² 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 SD² shows clear advantages. Across all datasets, SD² 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 SD² also degrades in performance on these tasks compared to UltraChat or similar tasks, it can consistently match or beat both pretrained

Acknowledgments

We thank Benjamin Estermann for his valuable input and discussions during the early stages of this project.

References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. GPT-4 Technical Report. arXiv:2303.08774.
- Ankner, Z.; Parthasarathy, R.; Nrusimha, A.; Rinard, C.; Ragan-Kelley, J.; and Brandon, W. 2024. Hydra: Sequentially-Dependent Draft Heads for Medusa Decoding. In *Proceedings of the Conference on Language Modeling (CoLM)*.
- Burton, F. W. 2012. Speculative Computation, Parallelism, and Functional Programming. *IEEE Transactions on Computers*, 100(12): 1190–1193.
- Cai, T.; Li, Y.; Geng, Z.; Peng, H.; Lee, J. D.; Chen, D.; and Dao, T. 2024. MEDUSA: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Chalnev, S.; Siu, M.; and Conmy, A. 2024. Improving Steering Vectors by Targeting Sparse Autoencoder Features. arXiv:2411.02193.
- Chen, C.; Borgeaud, S.; Irving, G.; Lespiau, J.-B.; Sifre, L.; and Jumper, J. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. arXiv:2302.01318.
- Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.
- Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168.
- Dettmers, T.; Lewis, M.; Belkada, Y.; and Zettlemoyer, L. 2022. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.
- Ding, N.; Chen, Y.; Xu, B.; Qin, Y.; Hu, S.; Liu, Z.; Sun, M.; and Zhou, B. 2023. Enhancing Chat Language Models by Scaling High-Quality Instructional Conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Du, C.; Jiang, J.; Xu, Y.; Wu, J.; Yu, S.; Li, Y.; Li, S.; Xu, K.; Nie, L.; Tu, Z.; et al. 2024. GliDe with a CaPE: A Low-Hassle Method to Accelerate Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Dubois, Y.; Galambosi, B.; Liang, P.; and Hashimoto, T. B. 2024. Length-Controlled AlpacaEval: A Simple Way to Debias Automatic Evaluators. arXiv:2404.04475.
- Fu, Y.; Bailis, P.; Stoica, I.; and Zhang, H. 2024. Break the Sequential Dependency of LLM Inference Using Lookahead Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Ge, T.; Xia, H.; Sun, X.; Chen, S.-Q.; and Wei, F. 2022. Lossless Acceleration for Seq2seq Generation with Aggressive Decoding. arXiv:2205.10350.
- Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.
- Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531.
- Hu, Y.; Liu, Z.; Dong, Z.; Peng, T.; McDanel, B.; and Zhang, S. Q. 2025. Speculative Decoding and Beyond: An In-Depth Survey of Techniques. arXiv:2502.19732.
- Huang, K.; Guo, X.; and Wang, M. 2024. SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths. arXiv:2405.19715.
- Huben, R.; Cunningham, H.; Smith, L. R.; Ewart, A.; and Sharkey, L. 2024. Sparse Autoencoders Find Highly Interpretable Features in Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Leviathan, Y.; Kalman, M.; and Matias, Y. 2023. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024a. EAGLE-2: Faster Inference of Language Models with Dynamic Draft Trees. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2024b. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Li, Y.; Wei, F.; Zhang, C.; and Zhang, H. 2025. EAGLE-3: Scaling Up Inference Acceleration of Large Language Models via Training-Time Test. arXiv:2503.01840.
- Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; Lu, C.; Zhao, C.; Deng, C.; Zhang, C.; Ruan, C.; et al. 2024a. DeepSeek-V3 Technical Report. arXiv:2412.19437.
- Liu, X.; Hu, L.; Bailis, P.; Cheung, A.; Deng, Z.; Stoica, I.; and Zhang, H. 2024b. Online Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Meta AI. 2024a. Introducing Llama 3.1: Our Most Capable Models to Date. <https://ai.meta.com/blog/meta-llama-3-1/>. Accessed: 2025-07-29.
- Meta AI. 2024b. 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/>. Accessed: 2025-07-29.
- Miao, X.; Oliaro, G.; Zhang, Z.; Cheng, X.; Wang, Z.; Zhang, Z.; Wong, R. Y. Y.; Zhu, A.; Yang, L.; Shi, X.; et al. 2024. 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)*.
- Narayan, S.; Cohen, S. B.; and Lapata, M. 2018. 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)*.
- OpenAI. 2024. Predicted Outputs Guide. <https://platform.openai.com/docs/guides/predicted-outputs>. Accessed: 2025-07-29.

- Park, K.; Choe, Y. J.; and Veitch, V. 2024. The Linear Representation Hypothesis and the Geometry of Large Language Models. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- Rimsky, N.; Gabrieli, N.; Schulz, J.; Tong, M.; Hubinger, E.; and Turner, A. 2024. Steering Llama 2 via Contrastive Activation Addition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Samragh, M.; Kundu, A.; Harrison, D.; Nishu, K.; Naik, D.; Cho, M.; and Farajtabar, M. 2025. Your LLM Knows the Future: Uncovering Its Multi-Token Prediction Potential. arXiv:2507.11851.
- ShareGPT. 2023. ShareGPT. https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered. Accessed: 2025-07-29.
- Shazeer, N. 2020. GLU Variants Improve Transformer. arXiv:2002.05202.
- Spector, B. F.; and Re, C. 2023. Accelerating LLM Inference with Staged Speculative Decoding. arXiv:2308.04623.
- Stern, M.; Shazeer, N.; and Uszkoreit, J. 2018. Blockwise Parallel Decoding for Deep Autoregressive Models. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.
- Sun, X.; Ge, T.; Wei, F.; and Wang, H. 2021. Instantaneous Grammatical Error Correction with Shallow Aggressive Decoding. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Sun, Z.; Mendlovic, U.; Leviathan, Y.; Aharoni, A.; Beirami, A.; Ro, J. H.; and Suresh, A. T. 2024. Block Verification Accelerates Speculative Decoding. arXiv:2403.10444.
- Sun, Z.; Suresh, A. T.; Ro, J. H.; Beirami, A.; Jain, H.; and Yu, F. 2023. SpecTr: Fast Speculative Decoding via Optimal Transport. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.
- Turner, A. M.; Thiergart, L.; Leech, G.; Udell, D.; Vazquez, J. J.; Mini, U.; and MacDiarmid, M. 2023. Steering Language Models with Activation Engineering. arXiv:2308.10248.
- Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.
- Xia, H.; Ge, T.; Wang, P.; Chen, S.-Q.; Wei, F.; and Sui, Z. 2023. Speculative Decoding: Exploiting Speculative Execution for Accelerating Seq2seq Generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 Technical Report. arXiv:2505.09388.
- Yin, M.; Chen, M.; Huang, K.; and Wang, M. 2024. A Theoretical Perspective for Speculative Decoding Algorithm. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.
- Zhang, L.; Wang, X.; Huang, Y.; and Xu, R. 2025. Learning Harmonized Representations for Speculative Sampling. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.
- Zhou, Y.; Lyu, K.; Rawat, A. S.; Menon, A. K.; Rostamizadeh, A.; Kumar, S.; Kagy, J.-F.; and Agarwal, R. 2024. DistillSpec: Improving Speculative Decoding via Knowledge Distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Zimmer, M.; Gritta, M.; Lampouras, G.; Ammar, H. B.; and Wang, J. 2025. Mixture of Attention for Speculative Decoding. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

A Technical Appendix

A.1 Notation

We use the following notation throughout the paper:

- 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 Training & Initialization 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). Regarding initialization, we set W_s as the 0 matrix, which means that the drafter’s behavior is initially unchanged by the noisy steering vector. We add a layer norm right after the computation of the steering vector, to stabilize the steering mechanism. We select the h, m, l layers inspired by EAGLE-2 choosing layers 3, $L/2, L - 2$

A.3 Ablation Studies

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. 7, these two methods only differ slightly, with SD^2 having the slight edge.

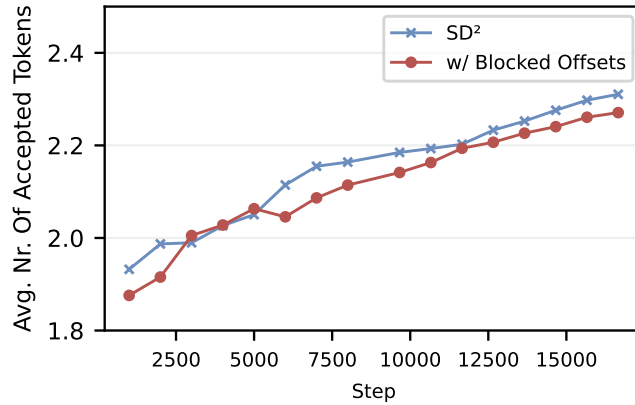


Figure 7: **Block efficiency throughout training with different training offset methods:** The training process of SD^2 uses a constant offset, i.e g_t is used for the prediction of $\hat{y}_{t+\delta}$, for training, where $\delta \in [1, k]$ random. We compare with *blocked offsets*, as they mimic the behavior observed during inference, by using g_t for the prediction of $\hat{y}_{t+1:t+k}$ if $t \bmod k = \delta - 1$. SD^2 slightly beats the blocked offset method. Here, the verifier used is Vicuna 1.3 7B, and Llama 160M is the drafter.

A.4 Statistical Significance

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

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

Table 3: Pairwise t-tests on block efficiency 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 SD². 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.

A.5 Example of speculative decoding

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

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- \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- ** ad- jacent** in the list, and \n- Use a **lin- ear 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 num- bers** with the threshold. \n \n - If any such pair has elements that the difference between them is **less than the ** differ- ence 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 num- bers 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 dis- tance** between any two sorting the list**, and then checking for elements the pairs later based on their element and its ** distance be- tween neighboring elements**, since saying that difference between each pair of consecu- tive elements**. \n\n ### Key Steps: \n\n1. **Sort the list ** so that the smallest num- ber comes first - This helps in easily find- ing neighboring elements. \n Sorting the ar- ray 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. \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 deter- mining 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 fol- lows: \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 compar- ing all pairs (because after sorting, the in- dices of elements that closest values to each other (i.e., pair will always be the second and second to of numbers will be consecutive el- ements in the sorted adjacent in the origi- nal list. \n - Traverse the \n - For each ele- ment in the sorted list Then , iterate through the sorted list and check if the difference be- tween the current number and the any two ad- jacent numbers is less than the given ‘ thresh- old’. \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\ndef 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 thresh- old. \n \n >>> has the given threshold. \n\n >>> has.close.elements (l than given thresh- old. \n \n >>> has float ‘given threshold. \n >>> has.close \n

Table 4: Continuation of qualitative example shown in Table 2. The text is generated using different drafting methods with $T = 1$ 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.