BLOCKPILOT: INSTANCE-ADAPTIVE POLICY LEARNING FOR DIFFUSION-BASED SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates inference by using a draft model to generate candidate tokens in parallel, which are then verified by the target model, enabling lossless acceleration. Recently, diffusion-based speculative decoding further improves parallelism by generating multiple tokens per forward pass via block-level diffusion, achieving state-of-the-art (SOTA) performance. However, existing methods adopt a fixed inference block size and assume a uniform optimal decoding strategy across all inputs. In this paper, we show that this assumption is suboptimal, as the optimal block size varies across samples and plays a critical role in speculative decoding performance. Moreover, these values exhibit a clear local structure, concentrating around the training block size, which reduces the problem to a low-dimensional and structured decision space. Based on these insights, we propose BlockPilot, a sample-adaptive block size selection method. Specifically, we formulate block size selection as a policy learning problem and propose an instance-adaptive decision mechanism that predicts the optimal block size based on the representation of the prefill stage. The prediction is performed only once after prefill. Extensive experiments demonstrate that BlockPilot consistently improves efficiency, achieving an acceptance length of 5.92 and a 4.20$\times$ speedup on Qwen3-4B under temperature $T = 1$.

1 INTRODUCTION

Large Language Models (LLMs) Touvron et al. (2023); Yang et al. (2025); Chiang et al. (2023) have achieved remarkable performance across a wide range of tasks (Annepaka & Pakray, 2025; Hadi et al., 2023), demonstrating strong capabilities in reasoning, code generation, and open-ended dialogue. Despite these advances, their inference efficiency is still fundamentally constrained by token-by-token autoregressive decoding Tay et al. (2022); Papa et al. (2024); Wan et al. (2023). Since each token must be generated conditioned on previously produced tokens, the decoding process is inherently sequential, leading to high latency and limited parallelism, especially for long-form generation. To alleviate this bottleneck, speculative decoding Leviathan et al. (2023); Li et al. (2025c; 2024; 2025b); Cai et al. (2024) introduces a draft model to propose multiple candidate tokens ahead of the target model. These candidates are then verified by the target model in parallel, allowing multiple tokens to be accepted within a single decoding step. As a result, speculative decoding enables lossless acceleration without altering the output distribution of the target model.

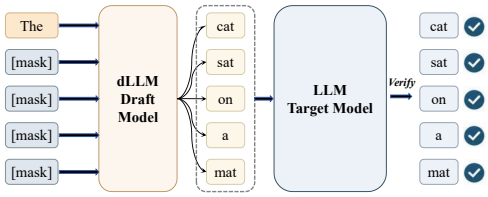


Figure 1: Diffusion-based speculative decoding with a dLLM draft model. The dLLM proposes a block of tokens in parallel, while the target LLM verifies the block and accepts the longest consistent prefix.

In recent years, diffusion-based language models (dLLMs) Nie et al. (2025); Arriola et al. (2025); Wu et al. (2025); Cheng et al. (2025) have emerged as a promising direction, and diffusion-driven speculative decoding further improves parallelism. By using a dLLM as the draft model, block-wise diffusion enables the generation of multiple tokens in a single forward pass, significantly reducing decoding latency and improving hardware utilization. Fig. 1 illustrates the paradigm of speculative decoding based on diffusion models. However, early approaches Li et al. (2025a); Sandler et al.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

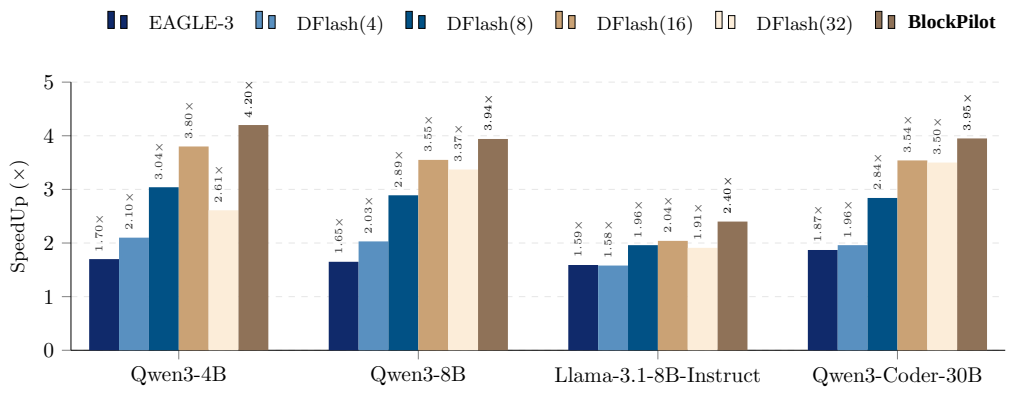


Figure 2: Speedup comparison across models under temperature $T = 1$. BlockPilot achieves the highest acceleration across all settings. Here, DFlash(n) denotes DFlash with block size n.

(2025); Samragh et al. (2025) rely on large diffusion models and exhibit weak coupling with the target model, limiting their practicality. More recently, DFlash Chen et al. (2026) represents a pioneering and influential advance toward practical block diffusion-based speculative decoding. By injecting hidden representations from the target model into the diffusion draft model, it enables high-quality parallel drafting and substantially improves acceptance length and real-world speedup.

Although this paradigm offers strong potential for parallelism, existing methods typically use a fixed block size during inference, directly inherited from the training stage. This design is simple and easy to deploy; however, it overlooks a critical dimension: the decoding policy. Treating one block size as uniformly optimal can be suboptimal when the best measured degree of parallelism varies across inputs. Differences in semantic constraints, contextual determinism, and token-level predictability provide a plausible source of varying tolerance for parallel drafting. Larger blocks can improve efficiency when prefix survival remains high, but they may lower acceptance when drafting errors accumulate; smaller blocks are more conservative but may underuse the parallel capacity of diffusion-based generation. The empirical sweep below tests this sample-dependence directly rather than assuming a universal relationship between prompt properties and block size.

In this paper, we revisit diffusion-based speculative decoding from a largely overlooked perspective. Beyond designing stronger draft models or more efficient verification mechanisms, we ask whether the decoding strategy itself should be treated as a learnable component. We treat block size as a control variable that influences both the available draft horizon and prefix-survival behavior. To evaluate this view, we conduct a systematic block-size sweep across multiple representative datasets. The results show a clear mismatch between the measured sample-wise optima and the fixed configuration used during training. Only a subset of samples achieve their highest measured acceptance length under the default setting, while many prefer different inference-time block sizes. This suggests that fixed strategies leave part of the available efficiency gain unrealized in the evaluated settings.

Furthermore, we observe that although the optimal block size varies across samples, its distribution is not arbitrarily scattered. Instead, it exhibits a clear local structure: for most samples, the optimal block size concentrates within a narrow region around the training configuration, and few samples achieve optimal performance outside this region. This locality has an important methodological implication. It transforms what would otherwise require expensive online search into a small-scale, discrete, and well-structured classification problem that can be handled by a predictor.

Based on this insight, we propose BlockPilot, a sample-adaptive block size selection method. Specifically, after the target model completes prefill, we use the raw logits at the final position as a representation of the current decoding state. Since the final position has aggregated full-context information via autoregressive attention, its logits reflect contextual constraints and token-level confidence, serving as an effective signal for predicting the acceptable block length. Therefore, we formulate block size selection as a policy learning problem over a discrete local action space and approximate the optimal policy with a predictor. The prediction is performed only once after prefill and does not modify the target model, draft model, or verification process. Extensive experiments across models and datasets demonstrate that BlockPilot further improves the efficiency of speculative decoding

without altering the original inference framework, achieving an acceptance length of 5.92 and a 4.20× speedup on Qwen3-4B at a temperature of 1. Fig. 2 presents the speedup comparison across different methods. Our contributions can be summarized as follows:

- We identify decoding policy as a learnable component in diffusion-based speculative decoding, showing that fixed block-size strategies are suboptimal across diverse inputs.
- We show empirically that the measured optimal block size follows a structured local distribution, enabling policy learning over a discrete action space.
- We propose BlockPilot, an instance-adaptive policy that predicts block size from the prefill state and achieves consistent speedup gains.

2 METHODOLOGY

2.1 PROBLEM FORMULATION

Speculative decoding based on diffusion language models Li et al. (2025a); Sandler et al. (2025); Samragh et al. (2025); Chen et al. (2026) is an efficient framework for accelerating autoregressive inference. It trains a diffusion language model as the draft model with a block size of B, and leverages a block-level diffusion mechanism to generate B tokens in parallel. The target autoregressive model then verifies the generated sequence in parallel, alleviating the sequential bottleneck of autoregressive decoding. Formally, within a single speculative decoding step, the average per-token generation latency is defined as follows Chen et al. (2026):

$$L(B) = \frac{T_{\text{draft}}(B) + T_{\text{verify}}(B)}{\tau(B)} \tag{1}$$

where $L(B)$ denotes the average token generation latency under block size B, $T_{\text{draft}}(B)$ and $T_{\text{verify}}(B)$ represent the computational costs of the draft generation and verification stages, respectively, and $\tau(B)$ denotes the expected number of accepted tokens per verification step. Furthermore, the end-to-end speedup $\eta(B)$ is defined as:

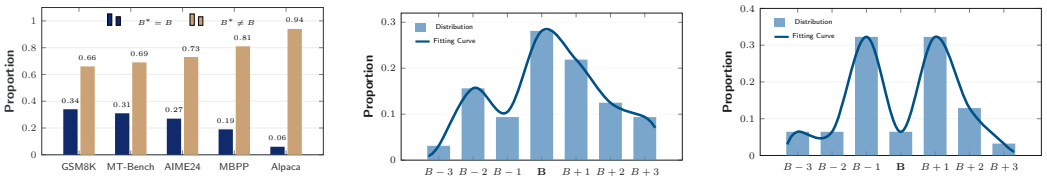
$$\eta(B) = \frac{L_{\text{AR}}}{L(B)} \tag{2}$$

where L_{AR} denotes the average latency of standard autoregressive decoding. Draft generation and verification process the tokens within a block in parallel, so their latency need not grow in direct proportion to B Hooper et al. (2025); Stern et al. (2018); Santilli et al. (2023). In our measured block-size range, the combined draft–verify latency varies within a narrow interval, making $\tau(B)$ the principal factor that distinguishes the resulting per-token latency. This observation motivates selecting B^* by maximizing acceptance length; Appendix E reports the latency measurements and verifies that acceptance-length and end-to-end-speedup criteria identify the same optimum in the evaluated setting.

In common settings, inference typically uses the same block size B as in training to maintain consistency between training and inference. However, this choice may be suboptimal at inference time because the best block size can vary across input samples and deviate from the training configuration. We therefore use the maximizer of expected acceptance length as the population block-selection oracle for each sample. This objective is not algebraically identical to maximizing speedup when draft–verify cost varies with block size; rather, it is justified in our evaluated regime by the narrow measured cost variation and by the observed agreement between the acceptance-length and end-to-end-speedup maximizers reported in Appendix E.

2.2 KEY FINDINGS

To systematically analyze the impact of block size on speculative decoding performance, we perform an exhaustive sweep over candidate block sizes on multiple representative datasets. This allows us to directly compare decoding behavior under different levels of block-level parallelism. Specifically, we define the global analysis set as $\mathcal{B}_{\text{glob}} = \{1, 2, \ldots, 2B\}$, where B denotes the block size used



(a) Proportion of samples with $\widehat{B}^*_{\text{glob}}$ matching or mismatching B.

(b) Unimodal distribution of B^*_{glob} peaking at the trained size B.

(c) Symmetric bimodal distribution of $\widehat{B}^*_{\text{glob}}$ near the trained size B.

Figure 3: Analysis of the empirical global-sweep optimum $\widehat{B}^*_{\text{glob}}$. (a) Matching and mismatching proportions across datasets. (b-c) Empirical distribution patterns, where the range $[B-3, B+3]$ covers the measured global-sweep optimum for nearly all evaluated samples.

during training. For each input sample x, we define its global-sweep optimum as follows:

$$\widehat{B}^*_{\text{glob}}(x) = \text{TB}\left(\arg\max_{b \in \mathcal{B}_{\text{glob}}} \widehat{\tau}(b; x)\right) \tag{3}$$

where $\widehat{\tau}(b; x)$ denotes the measured average number of accepted tokens for sample x under block size b, and TB selects the highest-priority maximizer under a fixed total ordering whenever multiple candidates attain the same maximum. The acceptance length determines the effective number of generated tokens per decoding step and therefore directly reflects how efficiently the draft tokens are utilized. It thus serves as the metric for constructing the sample-optimal block-size labels. Appendix E provides further discussion of the rationale for using acceptance length as the evaluation metric for block-size selection, while Appendix A distinguishes this empirical optimum from its population counterpart.

Finding I: Instance-wise Variability of Optimal Block Size. We first observe that the sample-wise empirical optimum $\widehat{B}^*_{\text{glob}}(x)$ varies significantly across samples and does not necessarily match the fixed block size B used during training. As shown in Fig. 3a, we report the proportions of samples satisfying $\widehat{B}^*_{\text{glob}}(x) = B$ and $\widehat{B}^*_{\text{glob}}(x) \neq B$ for each dataset. The results indicate that only a subset of samples align with the training configuration, while a substantial fraction prefer different block sizes at inference time, and this pattern is consistent across datasets.

These observations suggest that a fixed block size is insufficient to capture the best measured decoding behavior across diverse inputs. One plausible explanation is variation in context structure and predictability: strongly constrained inputs may sustain prefix survival over larger blocks, whereas less predictable continuations may accumulate drafting errors more rapidly as block size increases. This interpretation is consistent with the prefix-survival analysis in Appendix A, but the sweep itself establishes association rather than a causal decomposition of input properties. The mismatch between the fixed training-time block size and the sample-specific empirical optimum motivates adaptive block-size selection at inference time.

Finding II: Local Interval Property of Optimal Block Size. Although the optimal block size varies across samples, its distribution exhibits a clear local interval structure. As shown in Fig. 3, we observe two dominant patterns in the distribution, both indicating strong locality. Fig. 3b shows a unimodal distribution peaked at the training block size B, with probabilities rapidly decaying on both sides. Fig. 3c instead presents a bimodal but still localized pattern, with both peaks near B. In the global sweep, nearly all evaluated samples satisfy

$$\widehat{B}^*_{\text{glob}}(x) \in \{B-3, \ldots, B+3\}. \tag{4}$$

Once the block size deviates from B beyond this observed range, the acceptance length drops sharply, making such choices rarely optimal in the evaluated sweep. This indicates that the optimal block size exhibits strong empirical locality, with observed solutions concentrated within a narrow region around the training block size. Appendix A.4 analyzes the mechanisms behind the predominantly unimodal and approximately symmetric bimodal patterns, while Appendix F reports

their dataset-level proportions. Although $k = 3$ covers nearly all observed optima, we use $k = 2$ by default because it provides a better empirical trade-off between candidate coverage and prediction difficulty, as shown in Table 4.

This locality has important methodological implications. It reduces the broad global candidate set to a small discrete interval that retains most observed optima, substantially simplifying the prediction problem. More importantly, it enables learning-based strategies that select block sizes from a limited candidate set rather than performing a global search.

Finding III: Classification Formulation of Block Size Selection. Building on Finding II, which reveals a strong local interval structure of the measured optimal block size around the training block size, we formulate sample-adaptive block-size selection as a classification task over a finite discrete space. Specifically, we restrict the candidate set to a local neighborhood $\{B - k, \ldots, B + k\}$ and learn a function that predicts the empirical local optimum conditioned on the input sample. We use the raw logits at the final prefill position as the input representation. Under causal autoregressive attention, this position can attend to the complete prefill context, and its logits encode the target model's vocabulary-level next-token scores conditioned on that context. These scores retain both relative token preferences and their unnormalized scale, providing a candidate signal for block-size selection rather than a theoretically sufficient statistic. The comparison in Appendix B empirically evaluates this design against alternative input signals. The proposed module is integrated into the speculative decoding framework without modifying the draft or verification procedure.

2.3 Training

Following the classification formulation of block-size selection, we learn a score function parameterized by θ:

$$f_\theta : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{B}_{\text{loc}}|}, \qquad o(x) = f_\theta(z(x)), \tag{5}$$

where $z(x) \in \mathbb{R}^{|\mathcal{V}|}$ denotes the raw logits at the final prefill position, $\mathcal{V}$ is the target-model vocabulary, and $o(x)$ contains one score for each candidate in $\mathcal{B}_{\text{loc}}$. The empirical optimum $\widehat{B}^*_{\text{loc}}(x)$ supplies the classification target. The training process consists of supervised data construction and learning the block-size predictor.

Supervised Data Construction. Since the optimal block size is not explicitly annotated, we construct supervised data via an offline enumeration strategy. For each input sample x, we first run the target model's prefill stage and extract the raw logits at the final position, denoted as $z(x)$, as the input feature. Then, over the local candidate set $\mathcal{B}_{\text{loc}} = \{B - k, \ldots, B + k\}$ determined by Finding II, we enumerate each candidate block size b and execute full speculative decoding to obtain the empirical acceptance length $\widehat{\tau}(b; x)$. The supervision signal is the empirical local optimum

$$\widehat{B}^*_{\text{loc}}(x) = \text{TB}\left(\arg\max_{b \in \mathcal{B}_{\text{loc}}} \widehat{\tau}(b; x)\right). \tag{6}$$

The relation between this finite-measurement label and the population local and global optima is analyzed in Appendix A. Based on this procedure, we construct the training dataset $\mathcal{D}$ as follows:

$$\mathcal{D} = \{(z(x_i), \widehat{B}^*_{\text{loc}}(x_i))\}_{i=1}^N \tag{7}$$

In practice, the candidate set contains $2k + 1$ choices, corresponding to five evaluations under the default $k = 2$. Evaluations across candidate block sizes are mutually independent and can be executed in parallel. This data construction procedure directly aligns with the optimization objective of speculative decoding, enabling the model to learn a mapping from decoding states to optimal decisions without relying on hand-crafted heuristics. The measured construction and training costs are reported in Appendix I.

Block Size Predictor. We adopt an n-layer multilayer perceptron as the policy network. The predictor directly maps the final-position raw logits $z(x)$ to scores over the local candidate set $\mathcal{B}_{\text{loc}}$. The output distribution is defined via a softmax over candidate block sizes:

$$P(b \mid x) = \frac{\exp(o_b(x))}{\sum_{b' \in \mathcal{B}_{\text{loc}}} \exp(o_{b'}(x))} \tag{8}$$

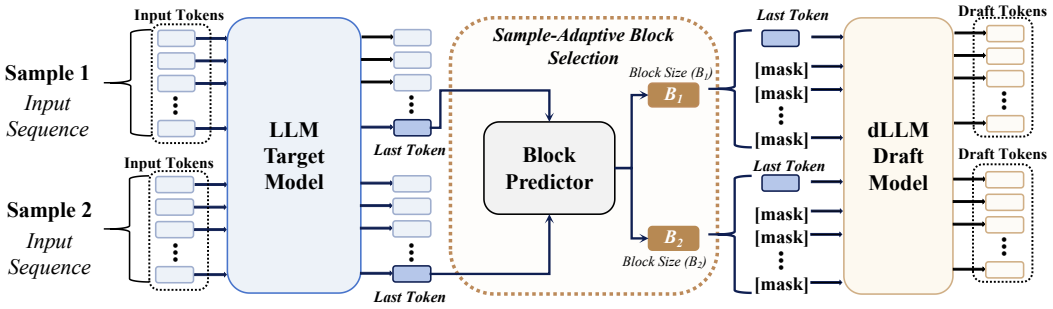


Figure 4: Overview of inference-stage sample-adaptive block size selection for diffusion-based speculative decoding. Given an input sequence, the target LLM performs prefill and produces the raw logits at the final position, which serve as a representation of the decoding state. These logits are then fed into the block size predictor to determine an instance-specific block size. Based on the predicted block size, the diffusion-based draft model generates a block of draft tokens in parallel.

where $o_b(x)$ denotes the predictor output logit corresponding to candidate block size $b \in \mathcal{B}_{\text{loc}}$ for input x. The model is trained by minimizing the standard cross-entropy loss over the training dataset:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N} \log P\Big(\widehat{B}^{*}_{\text{loc}}(x_i) \mid x_i\Big) \tag{9}$$

where N denotes the number of training samples and $\widehat{B}^{*}_{\text{loc}}(x_i)$ represents the measured local-optimal block-size label for sample x_i. This objective encourages the model to assign higher probability mass to the empirically optimal block size within the deployed candidate set.

2.4 Inference

During inference, the predictor enables sample-wise adaptive block size selection. Given an input sample x, we run the target model's prefill stage and extract the raw logits at the final position, denoted as $z(x)$. The predictor produces scores over the local candidate set $\mathcal{B}_{\text{loc}}$, and the block size is determined as follows:

$$\hat{B}(x) = \text{TB}\Big(\arg\max_{b \in \mathcal{B}_{\text{loc}}} o_b(x)\Big) \tag{10}$$

where $o_b(x)$ is the score assigned to candidate block size b. We apply the same fixed tie-breaking operator used during label construction when multiple candidates share the maximum score. Once obtained, $\hat{B}(x)$ remains fixed throughout subsequent draft generation and target-model verification. Fig. 4 presents the inference process, in which BlockPilot selects the level of parallel generation for each sample. Table 1 reports the parameter count, memory footprint, and inference latency of the predictor and backbone models. The predictor requires 7.34 ms and runs only once, immediately after prefill, so its latency does not accumulate across draft–verify iterations or reduce decoding-stage throughput. We include this one-time overhead in all end-to-end timing results, and it is amortized over the generated sequence and outweighed by the recurring latency reduction from the selected block size. This sample-specific policy decision distinguishes BlockPilot from fixed-block decoding while preserving decoding efficiency.

Table 1: Parameter count, memory usage, and inference latency of the block size predictor and backbone models.

Model	#Params	Memory		Latency
		Act.	Param.	
Qwen3-4B	4.41B	6.53 GB	8.62 GB	183.48 ms
Qwen3-8B	8.19B	8.16 GB	16.00 GB	278.37 ms
Llama-3.1-8B-Instruct	8.03B	7.10 GB	15.68 GB	272.14 ms
Block Size Predictor	0.32B	0.02 MB	0.62 GB	7.34 ms

3 Experiments

3.1 Experimental Setup

Models and Benchmarks. We evaluate our method on four representative LLMs: Qwen3-4B, Qwen3-8B Yang et al. (2025), Llama-3.1-8B-Instruct Grattafiori et al. (2024), and Qwen3-Coder-30B-A3B Cao et al. (2026), spanning diverse scales and domains, including both general-purpose

Table 2: Speedup ratios and average acceptance length τ on Qwen3 models across Math, Code, and Chat benchmarks. Q3-4B and Q3-8B denote Qwen3-4B and Qwen3-8B, respectively. DFlash(n) denotes DFlash with block size n. Additional results are provided in Appendix G.

Model	Method	Math						Code						Chat		Overall	
		GSM8K		MATH-500		AIME24		HumanEval		MBPP		SWE-Bench		MT-Bench		Avg.	
Temperature = 0		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Q3-4B	EAGLE-3	1.99×	3.30	1.83×	3.08	1.79×	3.05	1.84×	3.05	1.78×	2.95	1.39×	2.48	1.74×	3.02	1.70×	2.95
	DFlash (4)	2.29×	3.32	2.20×	3.48	2.12×	3.45	2.12×	3.22	2.16×	3.17	1.59×	2.58	1.45×	2.85	1.99×	3.15
	DFlash (8)	3.56×	5.16	3.59×	5.68	3.47×	5.64	3.19×	4.84	3.22×	4.71	1.95×	3.17	1.99×	3.91	2.99×	4.73
	DFlash (16)	4.55×	6.59	5.25×	8.31	4.58×	7.45	4.28×	6.51	4.26×	6.24	2.33×	3.78	2.69×	5.29	3.99×	6.31
	DFlash (32)	2.92×	4.24	3.22×	5.09	2.99×	4.86	2.56×	3.89	2.71×	3.97	1.69×	2.74	1.62×	3.19	2.53×	4.00
	BlockPilot	**4.76×**	**6.90**	**5.45×**	**8.62**	**4.80×**	**7.82**	**4.46×**	**6.77**	**4.38×**	**6.42**	**2.45×**	**3.98**	**2.85×**	**5.62**	**4.17×**	**6.59**
Q3-8B	EAGLE-3	1.94×	3.23	1.81×	3.02	1.79×	3.00	1.89×	3.17	1.69×	2.82	1.47×	2.45	1.63×	2.83	1.75×	2.93
	DFlash (4)	2.45×	3.29	2.50×	3.47	2.55×	3.44	2.45×	3.33	2.32×	3.21	1.96×	2.58	1.51×	2.58	2.25×	3.13
	DFlash (8)	3.76×	5.04	4.09×	5.67	4.22×	5.69	3.82×	5.19	3.51×	4.86	2.45×	3.23	2.01×	3.45	3.41×	4.73
	DFlash (16)	4.89×	6.56	5.98×	8.29	5.57×	7.51	4.98×	6.76	4.33×	5.99	2.72×	3.58	2.48×	4.25	4.42×	6.13
	DFlash (32)	4.47×	5.99	5.66×	7.84	5.20×	7.01	4.66×	6.32	3.92×	5.42	2.50×	3.29	2.37×	4.06	4.11×	5.70
	BlockPilot	**5.02×**	**6.73**	**6.17×**	**8.56**	**6.16×**	**8.30**	**5.21×**	**7.07**	**4.53×**	**6.26**	**2.96×**	**3.90**	**2.55×**	**4.37**	**4.66×**	**6.46**
Temperature = 1		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Q3-4B	EAGLE-3	1.89×	3.22	1.75×	2.99	1.64×	2.79	1.74×	3.01	1.69×	2.89	1.47×	2.31	1.77×	2.99	1.70×	2.88
	DFlash (4)	2.32×	3.17	2.34×	3.26	2.08×	2.90	2.29×	3.11	2.25×	3.10	1.79×	2.36	1.62×	2.74	2.10×	2.95
	DFlash (8)	3.60×	4.91	3.71×	5.16	2.97×	4.13	3.37×	4.58	3.21×	4.42	2.17×	2.86	2.24×	3.79	3.04×	4.26
	DFlash (16)	4.36×	5.95	4.83×	6.72	3.66×	5.10	4.32×	5.87	4.02×	5.54	2.33×	3.07	3.10×	5.23	3.80×	5.35
	DFlash (32)	2.83×	3.87	3.35×	4.66	2.79×	3.88	2.76×	3.76	2.69×	3.71	1.91×	2.52	1.91×	3.22	2.61×	3.66
	BlockPilot	**4.77×**	**6.51**	**5.41×**	**7.53**	**4.02×**	**5.60**	**4.74×**	**6.45**	**4.59×**	**6.33**	**2.58×**	**3.39**	**3.31×**	**5.59**	**4.20×**	**5.92**
Q3-8B	EAGLE-3	1.87×	3.12	1.73×	2.91	1.63×	2.74	1.75×	3.05	1.64×	2.74	1.33×	2.11	1.58×	2.70	1.65×	2.77
	DFlash (4)	2.34×	3.17	2.25×	3.22	2.05×	2.92	2.25×	3.05	2.23×	3.03	1.66×	2.20	1.44×	2.45	2.03×	2.86
	DFlash (8)	3.48×	4.72	3.53×	5.06	2.92×	4.17	3.26×	4.41	3.20×	4.36	1.94×	2.57	1.93×	3.28	2.89×	4.08
	DFlash (16)	4.28×	5.81	4.62×	6.62	3.49×	4.98	4.19×	5.68	3.98×	5.41	2.09×	2.77	2.20×	3.75	3.55×	5.00
	DFlash (32)	4.10×	5.57	4.41×	6.32	3.39×	4.84	3.77×	5.11	3.68×	5.00	2.01×	2.67	2.23×	3.79	3.37×	4.75
	BlockPilot	**4.69×**	**6.37**	**5.06×**	**7.25**	**3.84×**	**5.47**	**4.65×**	**6.30**	**4.54×**	**6.17**	**2.30×**	**3.05**	**2.48×**	**4.23**	**3.94×**	**5.55**

instruction-tuned and code-specialized models. We consider benchmarks across three task categories. For mathematical reasoning, we use GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2023), and AIME24 (MAA, 2025). For code generation and software engineering, we adopt HumanEval (Chen, 2021), MBPP (Austin et al., 2021), and SWE-Bench (Jimenez et al., 2023). For conversational generation, we use MT-Bench (Zheng et al., 2023). Together, these benchmarks cover Math, Code, and Chat scenarios.

Baselines. We compare our method with standard autoregressive decoding (baseline), the most classical speculative decoding method with an autoregressive drafter, EAGLE-3 (Li et al., 2025b), and the pioneering diffusion-based speculative decoding framework DFlash (Chen et al., 2026). Here, DFlash(n) denotes DFlash with block size n.

Metrics. Since our method preserves the exact output distribution of the target model under speculative decoding, generation quality remains unchanged. We therefore evaluate efficiency using average acceptance length τ, defined as the average number of tokens accepted per draft–verify cycle, and the speedup ratio, defined as the ratio of standard autoregressive decoding time to speculative decoding time.

Implementation Details. Our experiments are conducted on NVIDIA H100 80GB GPUs. We construct the training dataset from ShareGPT Chen et al. (2024), WSC Levesque et al. (2012), and COPA Roemmele et al. (2011). We train the model for 100 epochs using the Adam optimizer with a learning rate of $1e^{-5}$. The predictor network consists of two layers with a hidden dimension of 2048, and the default value of the hyperparameter k is set to 2. Further details on dataset construction and predictor training are provided in Appendix I. All speedup ratios are obtained from end-to-end wall-clock measurements that include the complete generation pipeline. We repeat each timing experiment multiple times under the same configuration and observe highly consistent run-to-run results; the reported speedup ratios are computed from the average measurements across these repeated runs.

3.2 Main Results

Table 2 presents results on Qwen3 models under deterministic and stochastic decoding. Although BlockPilot introduces an additional block-size predictor, it drives acceleration by converting the prefill representation into a sample-specific block-size decision, improving acceptance length and speedup beyond fixed-block configurations. These improvements require no modification to the draft or target models. Our method consistently achieves the best performance across models, temperatures, and benchmark categories. On Qwen3-4B, it reaches average speedups of 4.17× and 4.20× under temperature $= 0$ and temperature $= 1$, respectively. On Qwen3-8B, it achieves 4.66× and 3.94× under the two settings. These results outperform EAGLE-3 and all fixed-block DFlash variants, demonstrating the effectiveness of sample-adaptive block size selection. This indicates that the adaptive strategy generalizes across model scales and decoding regimes.

Compared with fixed-block DFlash(16), our method improves both speedup and average acceptance length τ. Under temperature $= 0$, BlockPilot increases the average speedup from 3.99× to 4.17× and τ from 6.31 to 6.59 on Qwen3-4B. On Qwen3-8B, the speedup rises from 4.42× to 4.66×, while τ improves from 6.13 to 6.46. Similar gains occur under temperature $= 1$: the average speedup increases from 3.80× to 4.20× on Qwen3-4B and from 3.55× to 3.94× on Qwen3-8B. These results reveal the limitation of fixed-block inference: larger blocks provide more parallelism, but do not always yield better acceleration. For example, DFlash(32) often underperforms DFlash(16), suggesting that overly large blocks may reduce acceptance due to accumulated drafting errors. In contrast, our method selects a suitable block size for each input sample, better balancing drafting parallelism and verification acceptance. BlockPilot remains effective across Math, Code, and Chat benchmarks. Gains persist at temperature $= 1$ despite more stochastic generation and challenging draft-token acceptance. This suggests that the raw logits at the final prefill position provide a useful signal for adaptive block size selection. Complete results, concurrent serving, feedback updates, latency, and prediction strategies appear in Appendices B–G.

3.3 Ablation Studies

We conduct ablation studies on Qwen3-4B at zero temperature to assess how predictor architecture, input preprocessing, and candidate interval radius affect performance.

Predictor Configuration. To further understand the impact of predictor design, we report results under different configurations in Table 3. Fixing the depth to $L = 2$, increasing the hidden size from $D = 1024$ to $D = 2048$ improves both speedup and acceptance length τ, while further scaling to $D = 4096$ does not improve the aggregate results. This suggests that a moderate hidden width is sufficient to capture the mapping from prefill logits to block size decisions. We then fix the hidden size to $D = 2048$ and vary the depth. Compared to a single-layer predictor, a two-layer model achieves better overall performance, indicating the benefit of moderate nonlinearity. Increasing the depth to $L = 3$ leads to gains on HumanEval but slight drops on GSM8K, yielding no clear overall advantage. Therefore, we adopt a two-layer MLP with hidden size 2048 as the default predictor.

Table 3: Speedup and acceptance length (τ) under different predictor configurations.

Setting	GSM8K		HumanEval		MT-Bench	
	Speedup	τ	*Speedup*	τ	*Speedup*	τ
Hidden width ($L = 2$)						
$D = 1024$	4.73×	6.86	4.41×	6.70	2.85×	5.62
$D = 2048$	4.76×	6.90	4.46×	6.77	2.85×	5.62
$D = 4096$	4.76×	6.90	4.46×	6.77	2.85×	5.62
Predictor depth ($D = 2048$)						
$L = 1$	4.71×	6.83	4.42×	6.72	2.84×	5.59
$L = 2$	4.76×	6.90	4.46×	6.77	2.85×	5.62
$L = 3$	4.65×	6.74	4.67×	7.10	2.85×	5.62

Candidate Interval Radius. Table 4 evaluates the candidate interval radius k, which defines the block-size search range $\{B - k, \ldots, B + k\}$ centered at the training block size B. This hyperparameter balances candidate coverage and prediction difficulty. A smaller radius ($k = 1$) restricts the search space, simplifying prediction but potentially missing better block-size choices. Increasing the radius to $k = 2$ improves both

Table 4: Effect of the candidate interval radius k on speedup and acceptance length (τ).

Setting	GSM8K		HumanEval		MT-Bench	
	Speedup	τ	*Speedup*	τ	*Speedup*	τ
$k = 1$	4.67×	6.77	4.44×	6.75	2.78×	5.47
$k = 2$	4.76×	6.90	4.46×	6.77	2.85×	5.62
$k = 3$	4.64×	6.73	4.39×	6.67	2.87×	5.66

speedup and acceptance length on GSM8K and HumanEval, and brings gains on MT-Bench, suggesting that a wider interval provides more useful candidates. When further expanded to $k = 3$, performance does not improve on most metrics, likely because the larger candidate set introduces more competing choices and increases prediction difficulty. Therefore, we set $k = 2$ as the default configuration, which balances candidate coverage and prediction difficulty.

Predictor Input Preprocessing. We further study how preprocessing the predictor input affects performance in Table 5. The predictor input consists of the raw logits at the final prefill position. Compared with directly using these logits, both normalization and softmax preprocessing consistently reduce speedup and acceptance length τ. This suggests that the original logit scale and vocabulary-level structure contain information relevant to block-size prediction, while additional preprocessing may alter or partially suppress this signal. Therefore, we use the unprocessed logits as the predictor input.

Table 5: Effect of predictor input preprocessing on speedup and acceptance length (τ).

Setting	GSM8K		HumanEval		MT-Bench	
	Speedup	τ	*Speedup*	τ	*Speedup*	τ
Softmax	4.55×	6.59	4.28×	6.51	2.69×	5.29
Normalization	4.64×	6.73	4.44×	6.75	2.83×	5.58
BlockPilot	**4.76×**	**6.90**	**4.46×**	**6.77**	**2.85×**	**5.62**

4 Related Work

Speculative Decoding. Speculative decoding improves the efficiency of large language model inference by alleviating the inherent sequential constraint of autoregressive generation. Early approaches Leviathan et al. (2023) introduce a draft model to generate candidate token sequences, which are then verified in parallel by a larger target model. Building on this idea, Medusa Cai et al. (2024) eliminates the need for an external draft model by equipping the base LLM with multiple prediction heads, enabling parallel candidate generation via a tree-attention mechanism. More recent work in the EAGLE family Li et al. (2025c; 2024; 2025b) further explores feature-level speculative decoding by leveraging intermediate hidden states of the target model. EAGLE-1 predicts future hidden-state distributions to improve token acceptance rates, while EAGLE-2 introduces adaptive drafting structures to better balance efficiency and accuracy. EAGLE-3 further improves training scalability and generalization across model sizes.

Diffusion-based Speculative Decoding. Recent work explores diffusion-based draft generation for speculative decoding. TiDAR Liu et al. (2025) combines diffusion and autoregressive objectives for parallel drafting, but still fails to achieve lossless generation quality. Another line of work adapts autoregressive models for diffusion-style drafting. Samragh et al. (2025) use a LoRA adapter to enable parallel drafting from implicit future-token signals, though with limited effectiveness. DiffuSpec Li et al. (2025a) and SpecDiff-2 Sandler et al. (2025) rely on large pre-trained diffusion LMs with search or alignment to improve acceptance, but require 7B-scale draft models, incurring substantial memory and latency overhead that hinders deployment. All the above methods remain difficult to apply efficiently in practice. Recently, DFlash Chen et al. (2026) makes a pioneering contribution to practical block diffusion-based speculative decoding by injecting target-model hidden states into the diffusion drafter, substantially improving draft quality, acceptance length, and inference speed. A detailed comparison between BlockPilot and prior adaptive length-selection strategies is provided in Appendix H.

5 Conclusion

In this paper, we revisit diffusion-based speculative decoding and identify block size as a key factor in inference efficiency. Across the evaluated sweeps, the measured optimal block size varies across samples but exhibits strong locality around the training configuration, reducing selection to a small structured decision problem. Based on this observation, we propose a sample-adaptive predictor that uses the raw logits at the final prefill position to select a block size from a local candidate set. The method is applied once per sample. Although the predictor is an additional component, it is central to the observed speedup improvement because it converts the prefill representation into an input-conditioned decoding policy. These results show that decoding policy, in addition to model architecture, materially affects inference efficiency in the evaluated settings.

AI USE STATEMENT

We used generative AI tools for language revision of text written by us. The assistance covered grammar, wording, clarity, and paragraph organization. We did not use these tools to formulate the research question, design BlockPilot, implement the method, plan the experiments, or interpret the results. The work began with our block-size sweeps, which showed that the measured best block size differs across inputs but is usually close to the training setting. On this basis, we defined a local candidate set and trained a predictor using the final prefill logits. We made the choice of method, conducted the experiments, and drew the conclusions ourselves. We reviewed all suggested revisions against the technical content, experimental setup, and reported results. Where a suggestion was imprecise, technically inaccurate, or went beyond the evidence, we changed or removed it. We also retained our original technical claims and did not rely on generated text as a source of factual evidence. We take full responsibility for the final manuscript, including passages that received language assistance.

ETHICS STATEMENT

This work follows the ICLR Code of Ethics. Our study focuses on improving the efficiency of language-model inference and does not involve human participants, human-subject experiments, or the collection of new personal data. Predictor training relies on publicly released datasets, while evaluation is conducted on established benchmarks covering mathematics, code generation, software engineering, and conversational tasks. We use these resources in accordance with their applicable licenses, terms of use, and intended purposes. Our work is limited to methodological and systems-level improvements for speculative decoding. BlockPilot changes the block size used during speculative decoding, while the target model continues to verify the drafted tokens using the standard verification procedure. The reported improvements therefore reflect gains in measured acceptance length and inference speed under the evaluated settings. They should not be interpreted as improvements in the underlying capabilities, factuality, robustness, or safety of the target language models. To support reproducibility and appropriate interpretation of the results, we explicitly specify the evaluated models, datasets, tasks, sampling temperatures, hardware configurations, and experimental settings, and limit our conclusions to the settings supported by these experiments.

REPRODUCIBILITY STATEMENT

We provide the information needed to examine and repeat the main experiments. Section 2 describes how block-size labels are obtained from candidate evaluations, why the candidates are restricted to a local interval, and how the predictor uses the final prefill logits. Section 3 gives the training setup, baseline methods, model and benchmark choices, hardware, and evaluation metrics. The appendix supplies further analysis of the measured optima, additional results, comparisons of predictor choices, and the offline cost and limitations of label construction. The training labels are based on finite measurements; they are empirical choices rather than exact population optima. We state this distinction because it affects how the predictor and its results should be interpreted. Acceptance length is used to construct labels, while the claimed acceleration is assessed separately through end-to-end wall-clock timing. We include the complete generation pipeline in timing and repeat measurements under the same configuration before reporting averages. The paper also reports results across different models, tasks, and sampling temperatures, including settings where the gain is smaller. These details allow readers to check the reported speedups and judge how the method may behave outside the tested conditions.

REFERENCES

Yadagiri Annepaka and Partha Pakray. Large language models: a survey of their development, capabilities, and applications. *Knowledge and Information Systems*, 67(3):2967–3022, 2025.

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models, 2025. URL https://arxiv.org/abs/2503.09573.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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, 2024. URL https://arxiv.org/abs/2401.10774.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, et al. Qwen3-coder-next technical report. *arXiv preprint arXiv:2603.00729*, 2026.

Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In *European Conference on Computer Vision*, pp. 370–387. Springer, 2024.

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation, 2025. URL https://arxiv.org/abs/2510.06303.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6, 2023.

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. *arXiv preprint arXiv:2110.14168*, 2021.

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. *arXiv preprint arXiv:2407.21783*, 2024.

Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea preprints*, 1(3):1–26, 2023.

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

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. *KR*, 2012(13th):3, 2012.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 19274–19286, 2023.

Guanghao Li, Zhihui Fu, Min Fang, Qibin Zhao, Ming Tang, Chun Yuan, and Jun Wang. Diffuspec: Unlocking diffusion language models for speculative decoding, 2025a. URL https://arxiv.org/abs/2510.02358.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024. URL https://arxiv.org/abs/2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test, 2025b. URL https://arxiv.org/abs/2503.01840.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty, 2025c. URL https://arxiv.org/abs/2401.15077.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression, 2025. URL https://arxiv.org/abs/2511.08923.

MAA. American Invitational Mathematics Examination - AIME, 2025. URL https://maa.org/math-competitions/american-invitational-mathematics-examination-aime.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models, 2025. URL https://arxiv.org/abs/2502.09992.

Lorenzo Papa, Paolo Russo, Irene Amerini, and Luping Zhou. A survey on efficient vision transformers: algorithms, techniques, and performance benchmarking. *IEEE transactions on pattern analysis and machine intelligence*, 46(12):7682–7700, 2024.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pp. 90–95, 2011.

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. URL https://arxiv.org/abs/2507.11851.

Jameson Sandler, Jacob K. Christopher, Thomas Hartvigsen, and Ferdinando Fioretto. Specdiff-2: Scaling diffusion drafter alignment for faster speculative decoding, 2025. URL https://arxiv.org/abs/2511.00606.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355, 2023.

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

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 2023.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm, 2025. URL https://arxiv.org/abs/2509.26328.

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. *arXiv preprint arXiv:2505.09388*, 2025.

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. *Advances in neural information processing systems*, 36:46595–46623, 2023.

A Formal Analysis of Sample-Adaptive Block Size Selection

A.1 Acceptance Length as a Prefix-Survival Process

Prefix-survival identity. We first align the random variable used in the analysis with the aggregation rule used to construct sample-level labels. For a prompt x and a fixed block size b, speculative decoding induces a sequence of draft–verify rounds whose states and number may depend on b and on decoding randomness. Consider the distribution over rounds induced by the evaluation protocol: if acceptance length is averaged within each generation and then across repeated generations, first sample a generation and then select one of its rounds uniformly; if all observed rounds are pooled, sample a round with the corresponding frequency weighting. Let $C_b(x)$ denote the number of consecutive draft tokens accepted in a round drawn from the aggregation rule actually used by the evaluation. We assume that every evaluated generation contains at least one such round. The population sample-level acceptance length is

$$\tau(b; x) = \mathbb{E}[C_b(x)]. \tag{11}$$

The expectation is over decoding randomness and the round distribution specified above. Thus, $\tau(b; x)$ is exactly the population counterpart of the empirical aggregation used in the block-size sweep. The survival identities below hold under either aggregation rule; the rule must simply remain fixed when candidates are compared.

Since $C_b(x)$ is a non-negative integer-valued random variable bounded by the block size b, its expectation can be decomposed into the sum of survival probabilities:

$$\tau(b; x) = \sum_{i=1}^{b} \mathbb{P}(C_b(x) \geq i). \tag{12}$$

This identity provides a prefix-level view of speculative decoding: each term measures the probability that the verified prefix survives up to at least position i.

In speculative decoding, verification accepts a prefix of the drafted block. Therefore, accepting at least i draft tokens is equivalent to the first i drafted tokens all passing the verification rule. Define the event

$$A_i^{(b,x)} = \{\text{the first } i \text{ drafted tokens are all accepted}\}. \tag{13}$$

Then

$$\mathbb{P}(C_b(x) \geq i) = \mathbb{P}(A_i^{(b,x)}). \tag{14}$$

Let $A_0^{(b,x)}$ denote the sure event. For every position whose preceding prefix has positive survival probability, define the conditional acceptance probability

$$q_j(x, b) = \mathbb{P}\Big(A_j^{(b,x)} \mid A_{j-1}^{(b,x)}\Big). \tag{15}$$

If $\mathbb{P}(C_b(x) \geq j - 1) = 0$, we set $q_j(x, b) = 0$ by convention. This convention makes the definition total and does not affect any survival probability, because every longer-prefix event then also has probability zero. By the chain rule of probability, the probability that the first i drafted tokens all survive verification is

$$\mathbb{P}(A_i^{(b,x)}) = \prod_{j=1}^{i} q_j(x, b). \tag{16}$$

Substituting this expression into the survival identity yields

$$\tau(b; x) = \sum_{i=1}^{b} \prod_{j=1}^{i} q_j(x, b). \tag{17}$$

Equation (17) shows that the sample-level acceptance length is the expected accepted-prefix length of a truncated prefix-survival process under the round distribution induced by block size b. The block size determines the truncation horizon, while each additional position contributes its probability of surviving verification. Because the induced generation trajectories can differ across block sizes, $q_j(x, b)$ includes both the block-dependent proposal behavior within a round and the distribution of verified-prefix states visited during generation.

Population quantities and empirical labels. The identities below concern the population quantity $\tau(b; x)$. In practice, the block-size sweep estimates it using a finite number of completed draft–verify rounds (and, for stochastic decoding, finite random seeds). Denote this estimate by $\widehat{\tau}(b; x)$. The empirical label can differ from the population optimum when candidates have nearly equal acceptance lengths. The formal comparisons remain exact for τ; applying them to $\widehat{\tau}$ additionally requires accounting for measurement error. If $|\widehat{\tau}(b; x) - \tau(b; x)| \leq \varepsilon_x$ for every enumerated candidate, then selecting the empirical maximizer incurs at most $2\varepsilon_x$ additional acceptance-length regret relative to the corresponding population maximizer. This follows directly by adding and subtracting the two estimation errors at the empirical and population optima.

Block-size-dependent acceptance. The conditional probability $q_j(x, b)$ should not be viewed as an intrinsic constant independent of the inference strategy. In block-level diffusion drafting, the draft model generates a set of mutually dependent tokens within a single block. Increasing the block size enlarges the maximum possible accepted length, but it may also make each prefix harder to verify because the proposal distribution must remain coherent over a longer span.

Therefore, increasing b induces two competing effects:

$$\tau(b; x) = \sum_{i=1}^{b} \underbrace{\prod_{j=1}^{i} q_j(x, b)}_{\text{prefix survival probability}} . \tag{18}$$

A larger block size increases the summation horizon by adding more candidate positions. However, the same change may alter the conditional acceptance probabilities inside each multiplicative term. Since prefix survival is multiplicative, even a mild degradation in $q_j(x, b)$ can be amplified over longer prefixes. Hence, the expected acceptance length is not determined solely by the number of drafted tokens, but by the interaction between block length and prefix survival.

A.2 Exact Comparison Between Block Sizes

The prefix-survival identity separates two effects of changing the block size without imposing a parametric form on the acceptance probabilities. Define

$$S_i(b; x) = \mathbb{P}(C_b(x) \geq i) = \prod_{j=1}^{i} q_j(x, b). \tag{19}$$

For two block sizes $b_2 > b_1$, subtracting their expected acceptance lengths gives the exact decomposition

$$\begin{aligned} \tau(b_2; x) - \tau(b_1; x) = & \underbrace{\sum_{i=1}^{b_1} [S_i(b_2; x) - S_i(b_1; x)]}_{\text{change in survival of shared prefix positions}} \\ & + \underbrace{\sum_{i=b_1+1}^{b_2} S_i(b_2; x)}_{\text{gain from the extended draft horizon}} . \end{aligned} \tag{20}$$

The second term is non-negative because a larger block offers additional positions that can be accepted. The first term can have either sign because changing the block size also changes the joint diffusion proposal and hence the survival probabilities at positions shared by both blocks. A larger block is preferable only when its horizon gain exceeds any loss in shared-prefix survival. Equation (20) therefore explains why acceptance length need not vary monotonically with block size and why the maximizing block size can differ across samples.

Importantly, Eq. (20) does not by itself imply that the optimum is centered near the training block size B. Establishing such a result would require additional assumptions about how $S_i(b; x)$ changes with the distance $|b-B|$, or direct measurements of these survival profiles. We therefore treat locality as an empirical finding from the global block-size sweep in Fig. 3, rather than as a consequence of an assumed retention function.

A.3 Empirical Locality and the Cost of Restricting the Candidate Set

Let $\mathcal{B}_{\text{glob}} = \{1, \dots, 2B\}$ denote the global sweep used for analysis. To handle ties explicitly, define the set of global maximizers

$$\mathcal{O}_{\text{glob}}(x) = \arg\max_{b \in \mathcal{B}_{\text{glob}}} \tau(b; x). \tag{21}$$

For radius k, the deployed local candidate set and its maximizers are

$$\mathcal{B}_{\text{loc}}(k) = \{b \in \mathcal{B}_{\text{glob}} : |b - B| \le k\}, \qquad \mathcal{O}_{\text{loc}}(x) = \arg\max_{b \in \mathcal{B}_{\text{loc}}(k)} \tau(b; x). \tag{22}$$

Let $B^*_{\text{glob}}(x) \in \mathcal{O}_{\text{glob}}(x)$ and $B^*_{\text{loc}}(x) \in \mathcal{O}_{\text{loc}}(x)$ denote representatives selected using the same deterministic tie-breaking rule as the label-construction procedure. The population coverage of the local interval is

$$c_k = \mathbb{P}(\mathcal{O}_{\text{glob}}(x) \cap \mathcal{B}_{\text{loc}}(k) \neq \emptyset)\,, \tag{23}$$

where the probability is taken over the prompt distribution. If population acceptance lengths were available, its finite-prompt estimate would be the fraction of prompts for which at least one global maximizer lies in the local interval. The actual sweep uses $\widehat{\tau}$ and therefore estimates this coverage subject to the measurement-error qualification above. Locality is thus measured rather than postulated, but its numerical estimate inherits finite-rollout uncertainty. On the population coverage event, the local and global oracle acceptance lengths are equal, regardless of which representatives are selected.

This observation yields a direct bound on the cost of restricting the action space. Define the truncation regret

$$\Delta_{\text{trunc}}(x) = \tau(B^*_{\text{glob}}(x); x) - \tau(B^*_{\text{loc}}(x); x). \tag{24}$$

Let $b_{\max} = \max \mathcal{B}_{\text{glob}}$. Since $0 \le \tau(b; x) \le b \le b_{\max}$,

$$0 \le \Delta_{\text{trunc}}(x) \le b_{\max}\, \mathbf{1}\{\mathcal{O}_{\text{glob}}(x) \cap \mathcal{B}_{\text{loc}}(k) = \emptyset\}\,. \tag{25}$$

Taking expectation gives

$$\mathbb{E}[\Delta_{\text{trunc}}(x)] \le b_{\max}(1 - c_k). \tag{26}$$

The bound is deliberately distribution-free and can be loose, but it makes the role of locality coverage explicit: high coverage limits the expected loss caused solely by pruning global candidates. An analogous finite-set inequality holds when the global and local maximizers, coverage, and regrets are all defined consistently from the same empirical scores $\widehat{\tau}$. The empirical truncation regret can then be computed directly from the exhaustive sweep.

A.4 Analysis of Unimodal and Bimodal Distribution Patterns

The two distribution patterns in Fig. 3 can be interpreted using the exact trade-off in Eq. (20). Increasing b extends the maximum draft horizon, but it can also change the shared-prefix survival probabilities because the diffusion drafter produces a different joint proposal. The balance between these two terms is sample-dependent.

For some inputs, shortening the block slightly can improve the conditional acceptance probabilities enough to compensate for the reduced horizon, leading to an optimum below B. For other inputs, prefix survival remains sufficiently high as the block grows, so the additional candidate positions dominate and the optimum moves above B. A third group attains the best balance at the training block size itself. When the responses of most samples to block-size changes are similar, their optima concentrate at or near B, producing the predominantly unimodal pattern in Fig. 3b.

The approximately symmetric bimodal pattern in Fig. 3c can arise when the evaluation set contains substantial groups on both sides of this trade-off. One group benefits from a slightly shorter block, whereas another can exploit a slightly longer block. Aggregating their sample-wise optima produces two modes around B, even though each sample is assigned a single optimal block size. Discrete candidate selection can sharpen this pattern because neighboring block sizes often have similar acceptance lengths, allowing modest sample-dependent changes in $q_j(x, b)$ to move the argmax from one side of B to the other.

This interpretation explains how unimodal and bimodal histograms can arise, but it does not prove why their modes should be close to the training block size. Their location is established by the

empirical sweep and may depend on the drafter training configuration, task composition, sampling temperature, and input uncertainty. The bimodal pattern should therefore be viewed as an aggregation of heterogeneous sample-level preferences rather than evidence of two intrinsic operating modes in the draft model.

A.5 Regret Decomposition for Local Block-Size Prediction

Acceptance-length regret. We finally separate the loss caused by restricting the candidate set from the loss caused by predictor error. Let $\hat{B}(x) \in \mathcal{B}_{\mathrm{loc}}(k)$ be the block size predicted by BlockPilot. Its regret relative to the global oracle is

$$R_{\mathrm{glob}}(x) = \tau(B^*_{\mathrm{glob}}(x); x) - \tau(\hat{B}(x); x). \tag{27}$$

Adding and subtracting the local-oracle acceptance length gives the exact decomposition

$$R_{\mathrm{glob}}(x) = \underbrace{\tau(B^*_{\mathrm{glob}}(x); x) - \tau(B^*_{\mathrm{loc}}(x); x)}_{\Delta_{\mathrm{trunc}}(x)} + \underbrace{\tau(B^*_{\mathrm{loc}}(x); x) - \tau(\hat{B}(x); x)}_{\Delta_{\mathrm{pred}}(x)}. \tag{28}$$

Thus, classification improvements affect only Δ_{pred}, whereas the interval radius k controls the coverage–difficulty trade-off through both terms.

For $k \geq 1$, define the sample-specific maximum adjacent variation on the finite ordered set $\mathcal{B}_{\mathrm{loc}}(k)$ as

$$\ell(x) = \max_{b, b+1 \in \mathcal{B}_{\mathrm{loc}}(k)} |\tau(b+1; x) - \tau(b; x)|. \tag{29}$$

By telescoping across adjacent block sizes, the prediction regret satisfies

$$0 \leq \Delta_{\mathrm{pred}}(x) \leq \ell(x) |B^*_{\mathrm{loc}}(x) - \hat{B}(x)|. \tag{30}$$

Unlike an assumed global Lipschitz constant, $\ell(x)$ is a well-defined population quantity for every sample on the finite candidate set and can be estimated from the sweep. Equation (30) shows that an adjacent classification error can be less costly than a distant one when the local acceptance profile is smooth. If the local set is a singleton, $\Delta_{\mathrm{pred}}(x) = 0$ and no variation constant is needed. A coarser bound follows from $\Delta_{\mathrm{pred}}(x) \leq b_{\max} \mathbf{1}\{\hat{B}(x) \neq B^*_{\mathrm{loc}}(x)\}$. If $e_{\mathrm{cls}} = \mathbb{P}(\hat{B}(x) \neq B^*_{\mathrm{loc}}(x))$ denotes population exact-match error under the specified tie-breaking rule, combining this inequality with Eq. (25) yields

$$\mathbb{E}[R_{\mathrm{glob}}(x)] \leq b_{\max} \left[(1 - c_k) + e_{\mathrm{cls}}\right]. \tag{31}$$

This result does not derive locality from the training block size. Instead, it states precisely how population locality coverage and predictor error jointly control regret relative to the global-sweep oracle. Let $\widehat{B}^*_{\mathrm{loc}}(x)$ be the empirical label constructed from $\widehat{\tau}$, let $e_{\mathrm{emp}} = \mathbb{P}(\hat{B}(x) \neq \widehat{B}^*_{\mathrm{loc}}(x))$, and let $e_{\mathrm{label}} = \mathbb{P}(\widehat{B}^*_{\mathrm{loc}}(x) \neq B^*_{\mathrm{loc}}(x))$. The union bound gives $e_{\mathrm{cls}} \leq e_{\mathrm{emp}} + e_{\mathrm{label}}$. Table 6 reports the finite-evaluation-set estimate of $1 - e_{\mathrm{emp}}$. Thus, interpreting the table as population-oracle accuracy additionally requires sufficiently stable acceptance-length measurements or an explicit estimate of label instability. Exact-match error may also be conservative under ties because predicting a different member of $\mathcal{O}_{\mathrm{loc}}(x)$ produces zero acceptance-length regret.

B Comparison of Block-Size Prediction Strategies

We further compare seven strategies for predicting the sample-specific block size using Qwen3-4B. All methods are evaluated as classifiers on the same constructed datasets. For each input x, we enumerate the candidate set $\mathcal{B}_{\mathrm{loc}} = \{B-2, B-1, B, B+1, B+2\}$ and use the block size with the largest measured acceptance length as the empirical label $\widehat{B}^*_{\mathrm{loc}}(x)$. We report exact-match classification accuracy, defined as the proportion of evaluation samples for which the predicted block size $\hat{B}(x)$ equals $\widehat{B}^*_{\mathrm{loc}}(x)$. BlockPilot uses the two-layer logit predictor described in the main text. The learnable predictors use the same training and evaluation splits, cross-entropy objective, Adam optimizer, learning rate of 1×10^{-5}, and 100 training epochs. The scalar thresholds used by the entropy and confidence baselines are fitted only on the corresponding training split and remain fixed during evaluation. This protocol changes only the predictor input or architecture and therefore isolates block-size prediction quality from the subsequent decoding process.

Entropy-based selection. This baseline uses no neural predictor and first converts the last-position raw logits into a probability distribution. Let $z(x) \in \mathbb{R}^{|\mathcal{V}|}$ denote the raw Qwen3-4B logits at the final prefill position. We obtain $p(x) = \mathrm{softmax}(z(x))$ and compute

$$H(x) = -\sum_{v \in \mathcal{V}} p_v(x) \log p_v(x). \tag{32}$$

Low entropy indicates a concentrated next-token distribution and is mapped to a larger block size, whereas high entropy is mapped to a smaller block size. Four ordered thresholds $\theta_1 < \theta_2 < \theta_3 < \theta_4$ partition the scalar entropy into five actions:

$$\hat{B}(x) = \begin{cases} B+2, & H(x) \le \theta_1, \\ B+1, & \theta_1 < H(x) \le \theta_2, \\ B, & \theta_2 < H(x) \le \theta_3, \\ B-1, & \theta_3 < H(x) \le \theta_4, \\ B-2, & H(x) > \theta_4. \end{cases} \tag{33}$$

We jointly search the four thresholds on the training split to maximize exact-match accuracy while preserving their order, and then apply the selected thresholds to every evaluation dataset without further fitting. This method uses an interpretable scalar rule, but the scalar entropy discards which tokens receive probability mass and cannot distinguish distributions with similar entropy but different semantic structures.

Maximum-probability selection. This baseline uses only the largest probability in the softmax distribution $p(x)$,

$$c_{\max}(x) = p_{(1)}(x) = \max_{v \in \mathcal{V}} p_v(x). \tag{34}$$

The value measures the confidence assigned to the most likely next token. We learn four ordered thresholds on the training split and map higher values of $c_{\max}(x)$ to larger block sizes. Specifically, the five intervals are assigned in ascending order to $\{B-2, B-1, B, B+1, B+2\}$. The thresholds are jointly selected to maximize training accuracy and are fixed for all evaluation datasets. Unlike entropy, this signal focuses only on the strongest prediction, but it ignores the probability assigned to competing tokens and the remainder of the vocabulary distribution.

Top-1–Top-2 margin selection. This baseline uses the probability margin between the two most likely tokens,

$$c_{\mathrm{margin}}(x) = p_{(1)}(x) - p_{(2)}(x), \tag{35}$$

where $p_{(1)}(x)$ and $p_{(2)}(x)$ are the largest and second-largest probabilities, respectively. A large margin indicates that the leading token is clearly separated from its closest alternative and is therefore mapped to a larger block size. As in the maximum-probability baseline, four ordered thresholds divide the margin into five intervals corresponding to $\{B-2, B-1, B, B+1, B+2\}$. This feature captures local ambiguity between the two leading candidates, but it still discards the identities of those candidates and all probability mass below the second-ranked token.

Top-10% sparse-logit predictor. This baseline examines whether retaining a larger set of high-confidence vocabulary coordinates is sufficient for block-size prediction. Let $K = \lceil 0.1|\mathcal{V}| \rceil$, and let $\mathcal{I}_K(x)$ contain the indices of the K largest entries in the last-position raw logits. We construct a sparse vector $\tilde{z}(x) \in \mathbb{R}^{|\mathcal{V}|}$ from $z(x)$ as

$$\tilde{z}_v(x) = \begin{cases} z_v(x), & v \in \mathcal{I}_K(x), \\ 0, & v \notin \mathcal{I}_K(x). \end{cases} \tag{36}$$

Unlike an input formed by sorting only the retained values, this construction preserves the original vocabulary coordinate of every selected token. The sparse vector is passed to the same two-layer MLP and five-way classification head used by BlockPilot, so the comparison isolates the effect of removing the lower-ranked 90% of vocabulary logits.

Table 6: Exact-match accuracy (%) of different block-size prediction strategies. Empirical labels are obtained by enumerating $\mathcal{B}_{\text{loc}} = \{B-2, \ldots, B+2\}$ and selecting the block size with the highest measured acceptance length.

Method	GSM8K	MATH-500	AIME24	HumanEval	MBPP	SWE-Bench	MT-Bench	Avg.
Entropy-based selection	63.4	60.8	61.7	66.9	65.7	60.3	60.6	62.8
Maximum-probability selection	66.1	64.8	63.9	68.7	67.8	61.9	62.5	65.1
Top-1–Top-2 margin selection	68.4	67.1	65.8	69.6	69.1	63.7	64.4	66.9
Top-10% sparse-logit predictor	69.2	68.0	66.7	70.1	69.6	64.8	65.4	67.7
Hidden-state predictor	92.6	91.8	90.9	93.4	92.9	88.8	89.7	91.4
Sparse MoE predictor	97.9	98.1	97.6	97.8	98.0	97.2	97.5	97.7
BlockPilot	**98.7**	**98.9**	**98.6**	**98.6**	**98.8**	**98.5**	**98.5**	**98.7**

Hidden-state predictor. This alternative examines whether the target model's contextual representation can replace its predictive output. For an input sequence of length L, Qwen3-4B produces final-layer prefill states $H(x) \in \mathbb{R}^{N\times L\times d_{\text{model}}}$. We select only the state at the final non-padding position, $h_L(x) \in \mathbb{R}^{N\times d_{\text{model}}}$, because it attends to the complete preceding context. No temporal pooling or vocabulary projection is applied. The selected state is passed through

$$u(x) = \text{GELU}(W_h h_L(x) + b_h), \qquad o(x) = W_o u(x) + b_o, \tag{37}$$

where $u(x) \in \mathbb{R}^{N\times 2048}$ and $o(x) \in \mathbb{R}^{N\times 5}$. The five output dimensions correspond to the same ordered block-size classes as the other methods. This representation is substantially lower-dimensional than the vocabulary logits and contains rich contextual information. However, it does not expose token-level predictive confidence as directly as the output layer.

Sparse MoE predictor. The mixture-of-experts (MoE) baseline tests whether substantially increasing predictor capacity improves block-size classification while retaining the same logit input as BlockPilot. Given a batch of raw last-position logits $Z \in \mathbb{R}^{N\times|\mathcal{V}|}$, a shared input layer first produces $h^{(0)} \in \mathbb{R}^{N\times 2048}$. The representation is then processed by three sparse MoE blocks. Each block contains eight experts, and each expert is a two-layer feed-forward network with dimensions $2048 \to 4096 \to 2048$ and a GELU activation. For block l, a learned router computes

$$g^{(l)} = \text{softmax}\Big(W_r^{(l)} h^{(l-1)}\Big). \tag{38}$$

Only the two experts with the largest routing probabilities are activated for each sample. Their routing weights are renormalized within the selected pair, and their outputs are combined through

$$h^{(l)} = \text{LayerNorm}\left(h^{(l-1)} + \sum_{e\in \text{Top2}(g^{(l)})} \tilde{g}_e^{(l)} E_e^{(l)}(h^{(l-1)})\right). \tag{39}$$

After three MoE blocks, a linear classification head maps $h^{(3)}$ to five output logits. The predicted class is converted to its corresponding block size in $\mathcal{B}_{\text{loc}}$. The vocabulary projection is shared by all experts, avoiding eight separate high-dimensional input layers. Thus, this baseline increases nonlinear capacity and allows expert specialization without changing the input signal or the output action space.

Results and analysis. As shown in Table 6, BlockPilot achieves an average exact-match accuracy of 98.7%, ranging from 98.5% to 98.9% across the seven benchmarks. The 0.4-point spread between its best and worst results is substantially smaller than the differences among task categories, indicating that its advantage is not confined to one benchmark type. The controlled comparisons attribute this result to two design choices: preserving the complete predictive output of the target model and using a nonlinear predictor whose capacity matches the small, structured action space.

The first advantage is information preservation. Entropy, maximum probability, and the Top-1–Top-2 margin achieve average accuracies of 62.8%, 65.1%, and 66.9%, respectively. Each compresses the vocabulary distribution into one scalar and then assumes a monotonic relation between confidence and block size. This compression removes token identity and conflates distributions that have the same scalar uncertainty but allocate probability mass to different alternatives. The Top-10% sparse-logit predictor retains token identities and a much larger set of scores, yet reaches only

67.7%, 31.0 points below BlockPilot. Its limited improvement shows that the highest-scoring coordinates alone are insufficient in this experiment. BlockPilot instead retains the complete logit vector, including the relative scores of the dominant token, its competitors, and lower-ranked vocabulary items. The observed gap is therefore consistent with block-size suitability depending on the joint shape of the predictive distribution rather than on a single confidence statistic or a truncated set of high-probability tokens.

The second advantage is representation–objective alignment. The hidden-state predictor already contains full-context information and raises average accuracy to 91.4%, but remains 7.3 points below BlockPilot. Unlike the hidden state, the last-position logits are the target model's explicit predictive output after the language-model head. They expose which tokens compete at the next position and their relative compatibility with the prompt, signals directly related to the uncertainty that the diffusion drafter must sustain over a block. Because these logits are already produced by prefill, BlockPilot uses this prediction-aligned representation without an additional sequence encoder or temporal pooling. The hidden-state comparison thus suggests that contextual richness alone does not explain the result; expressing that context in the target model's token-prediction space is also important.

Finally, the sparse MoE predictor tests whether BlockPilot's gain follows simply from insufficient baseline capacity. With the same full-logit input, the substantially larger MoE reaches 97.7% average accuracy, recovering most of the decision boundary but remaining 1.0 point below BlockPilot. The prediction problem contains only five neighboring block-size classes, so additional expert routing and depth do not provide a measurable advantage. In contrast, BlockPilot's two-layer MLP supplies enough nonlinearity to combine vocabulary-wide signals while optimizing a single shared decision function with all training samples. Taken together, the three comparison groups isolate complementary factors: scalar and sparse inputs lose predictive information, the hidden state is less directly aligned with token-level acceptance uncertainty, and the MoE adds capacity beyond what the local five-way decision requires. BlockPilot performs best because it combines the most informative evaluated input with a predictor of appropriate complexity, rather than because of either feature dimensionality or model size alone.

C Multi-Request Serving Performance with SGLang

We evaluate multi-request serving performance using SGLang on Qwen3-8B and AIME24 at temperature $T = 0$. The comparison includes EAGLE-3, fixed-block DFlash, and BlockPilot under request concurrency levels from 1 to 16. We report end-to-end speedup over standard autoregressive decoding executed with the same SGLang configuration and concurrency, so every entry includes request scheduling, batching, drafting, and verification in the complete serving pipeline.

Table 7 shows that BlockPilot achieves the highest speedup at every evaluated concurrency level, ranging from 4.99× with one active request to 3.68× at concurrency 16, and obtains the highest average speedup of 4.38×. All methods show an overall reduction in relative acceleration as concurrency increases, while the amount retained at each serving load varies modestly across methods and configurations. Under higher concurrency, continuous batching improves GPU utilization for autoregressive decoding and narrows the relative advantage of speculative execution. Concurrent requests can also occupy different generation stages, and their draft–verify workloads need not have identical execution shapes, reducing opportunities to form uniform batches. For BlockPilot, sample-specific block-size decisions introduce an additional source of shape diversity across simultaneously served requests. Even under these conditions, its adaptive policy consistently exceeds EAGLE-3 and every fixed-block DFlash configuration in the table. The result

Table 7: End-to-end speedup under concurrent SGLang serving on Qwen3-8B and AIME24 at temperature $T = 0$. For each concurrency level, autoregressive decoding under the same serving configuration is used as the baseline.

Method	Concurrency					Avg.
	1	2	4	8	16	
EAGLE-3	1.42×	1.35×	1.27×	1.25×	1.05×	1.27×
DFlash (4)	2.06×	1.90×	1.84×	1.74×	1.52×	1.81×
DFlash (8)	3.37×	3.17×	3.02×	2.92×	2.49×	3.19×
DFlash (13)	4.18×	3.85×	3.66×	3.55×	3.07×	3.66×
DFlash (14)	4.22×	4.02×	3.85×	3.62×	3.12×	3.77×
DFlash (15)	4.36×	4.02×	3.90×	3.80×	3.26×	3.87×
DFlash (16)	4.50×	4.18×	3.97×	3.82×	3.30×	3.95×
DFlash (17)	4.38×	4.18×	4.02×	3.84×	3.23×	3.93×
DFlash (18)	4.42×	4.08×	3.93×	3.75×	3.28×	3.89×
DFlash (19)	4.35×	4.09×	3.94×	3.80×	3.22×	3.88×
DFlash (32)	4.21×	3.94×	3.69×	3.58×	3.05×	3.69×
BlockPilot	**4.99**×	**4.57**×	**4.47**×	**4.20**×	**3.68**×	**4.38**×

Table 8: End-to-end speedup of one-shot block-size prediction and decoding-time feedback updating on representative Math, Code, and Chat benchmarks. The feedback variant reruns the block-size predictor after every draft–verify round using the raw logits at the last verified position.

Model	Temp.	GSM8K		HumanEval		MT-Bench	
		One-shot	Feedback	One-shot	Feedback	One-shot	Feedback
Qwen3-4B	0	4.76×	4.64×	4.46×	4.36×	2.85×	2.78×
Qwen3-4B	1	4.77×	4.66×	4.74×	4.61×	3.31×	3.23×
Qwen3-8B	0	5.02×	4.90×	5.21×	5.07×	2.55×	2.49×
Qwen3-8B	1	4.69×	4.58×	4.65×	4.53×	2.48×	2.42×
Llama-3.1-8B-Instruct	0	2.96×	2.89×	3.74×	3.65×	2.78×	2.71×
Llama-3.1-8B-Instruct	1	2.73×	2.67×	3.29×	3.21×	1.92×	1.87×
Qwen3-Coder-30B-A3B	0	3.75×	3.66×	6.05×	5.90×	2.49×	2.43×
Qwen3-Coder-30B-A3B	1	3.77×	3.68×	6.35×	6.18×	2.48×	2.42×

indicates that instance-conditioned block selection remains effective across multi-request serving loads, although the attainable relative gain is moderated by batching, scheduling, and synchronization effects at higher concurrency.

D One-Shot Prediction versus Periodic Feedback Updates

We compare the one-shot policy used by BlockPilot with a decoding-time feedback variant. BlockPilot executes the block-size predictor once immediately after target-model prefill and keeps the selected block size unchanged throughout the subsequent generation. In contrast, the feedback variant executes block-size prediction during every decoding iteration. After each draft–verify round, it extracts the raw logits at the last verified position, reruns the same predictor, and uses the resulting block size in the next round. The feedback variant therefore repeatedly incorporates the updated verified prefix and the latest decoding state into its decisions. The two strategies differ only in prediction frequency; the predictor architecture, target model, diffusion drafter, verification procedure, and local candidate set $\mathcal{B}_{\text{loc}}$ remain identical.

Table 8 shows that executing block-size prediction in every decoding iteration consistently yields lower end-to-end speedup than the one-shot policy across models, benchmarks, and decoding temperatures. Although the updated logits reflect the latest verified prefix, this additional information does not translate into higher overall efficiency. Repeated predictor execution adds latency after every draft–verify round. Repeated decisions can also change the drafting and verification shapes between consecutive rounds, introducing synchronization and configuration-switching costs. Moreover, an intermediate logit vector is strongly influenced by local next-token ambiguity, causing the selected block size to respond to short-term fluctuations even when the prefill decision remains appropriate for the sequence as a whole. The evaluated generations are not limited to short continuations: depending on the task and prompt, the outputs can span several hundred tokens and may exceed one thousand tokens. The consistent advantage of the one-shot policy over these decoding horizons indicates that the block size selected from prefill does not rapidly become obsolete as generation proceeds. Instead, the prefill logits capture sample-level properties that remain informative even as token-level uncertainty changes during a long output. Finally, the predictor is optimized for sample-level selection from the prefill logits, whereas repeated application during decoding exposes it to intermediate states that are less aligned with this objective. These effects outweigh the benefit of incorporating the latest verified prefix. Executing BlockPilot once after prefill and fixing the selected block size therefore provides a more stable and efficient decoding policy in the evaluated settings.

E Draft and Verification Latency Across Block Sizes

We measure the draft-model forward latency and target-model verification latency under different inference block sizes. All latency measurements in this experiment are conducted on Qwen3-4B with batch size 1 using an NVIDIA H100 80GB GPU, following the implementation setup used in the main experiments. The two stages are timed separately for each block size.

Table 9: Draft-model forward latency and target-model verification latency under different inference block sizes on Qwen3-4B. Both stages remain within narrow latency ranges as the number of jointly processed tokens increases from 2 to 32.

Block size	**2**	**4**	**8**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**32**
Draft latency (ms)	35.5	35.6	35.7	35.9	36.0	36.1	36.2	36.3	36.4	36.5	36.9
Verification latency (ms)	182.1	182.2	182.4	182.7	182.8	182.9	183.0	183.1	183.2	183.3	183.9

An appropriate labeling metric should measure the effective progress made by each draft–verify iteration and remain aligned with the end-to-end latency objective. Acceptance length satisfies the first requirement directly: $\tau(b; x)$ is the expected number of draft tokens that survive target-model verification for sample x under the specified aggregation protocol, and therefore measures the average verified progress of an iteration rather than how many tokens are merely proposed. It also permits a sample-level comparison among candidate block sizes without introducing a separate heuristic score.

Table 9 provides empirical support for the second requirement. Draft-model forward latency remains between 35.5 and 36.9 ms across all evaluated block sizes, while target-model verification remains between 182 and 184 ms. The combined draft–verify cost therefore changes from approximately 217.6 to 220.8 ms, a variation of less than 1.5%, even when the block size increases by a factor of 16 from 2 to 32. This behavior follows from the parallel structure of diffusion drafting and target verification: tokens within a block are processed concurrently, so increasing the number of jointly processed tokens does not increase latency proportionally. Kernel launches, model-weight access, and synchronization also contribute costs shared across the tokens in a block.

The average per-token latency can be written as $L(b) = [T_{\text{draft}}(b) + T_{\text{verify}}(b)]/\tau(b)$. Since the measured numerator varies within a narrow interval, the acceptance length provides the principal sample-dependent distinction among nearby candidate block sizes. A candidate that accepts more tokens amortizes a similar draft–verify cost over a longer verified prefix and consequently reduces per-token latency. Maximizing the measured acceptance length $\widehat{\tau}(b; x)$ is therefore well aligned with selecting the block size that improves inference efficiency, which supports its use in constructing $\widehat{B}^*_{\text{loc}}(x)$. We also compare the block size selected by maximum acceptance length with the one selected by maximum measured end-to-end speedup. Across the evaluated samples and candidate configurations, the two criteria identify exactly the same optimal block size, providing direct empirical evidence that acceptance length preserves the speedup-based optimum in our setting. Acceptance length is used to construct the supervised labels, while all reported system-level gains are independently evaluated using repeated end-to-end wall-clock measurements.

F Dataset-Level Distribution of Predicted Block Sizes

To complement the aggregate patterns in Fig. 3, we examine how the trained BlockPilot predictor selects block sizes on individual datasets. We use Qwen3-4B with temperature $T = 0$ and the default candidate set $\mathcal{B}_{\text{loc}} = \{14, 15, 16, 17, 18\}$, corresponding to the training block size $B = 16$ and interval radius $k = 2$. For each evaluation sample, the trained predictor selects one of the five candidates from the final-position raw logits produced during prefill. Table 10 reports the resulting prediction frequencies, normalized over all predictions within each dataset.

GSM8K and HumanEval show highly similar, center-dominant prediction distributions. In both cases, the trained predictor selects block size 16 most frequently, accounting for approximately 32% of the samples, while block size 17 forms the second-largest group at approximately 24–25%. The remaining predictions are distributed across both sides of the training configuration, showing that even within datasets sharing the same dominant choice, the learned policy assigns a substantial fraction of samples to a dif-

Table 10: Distribution (%) of block sizes predicted by the trained BlockPilot model on Qwen3-4B at temperature $T = 0$. Each row is normalized over predictions among the five candidate block sizes from 14 to 18.

Dataset	**14**	**15**	**16**	**17**	**18**
GSM8K	17.90	10.40	32.30	25.10	14.40
HumanEval	18.32	9.81	32.59	24.42	14.86
MT-Bench	6.70	34.00	6.70	38.00	14.60

ferent draft horizon. MT-Bench exhibits a markedly different pattern: the predictor assigns 34.0% and 38.0% of the samples to block sizes 15 and 17, respectively, while selecting block size 16 for only 6.7%. This bimodality indicates that the learned policy distinguishes between conversational samples that favor slightly shorter and slightly longer blocks instead of collapsing to the training configuration. Despite their different shapes, all three prediction distributions remain confined to the same narrow neighborhood of the training block size. The trained predictor therefore preserves the local structure established in our analysis while making nontrivial, sample-dependent choices within that structured action space.

G Complete Performance Results Across Models and Block Sizes

Tables 11 and 12 provide the complete speedup and acceptance-length results across all evaluated models, temperatures, benchmarks, and fixed DFlash block sizes. The first table expands the Qwen3-4B and Qwen3-8B results around the training block size, while the second reports the corresponding evaluation on Llama-3.1-8B-Instruct and Qwen3-Coder-30B-A3B. Reporting the intermediate block sizes makes it possible to distinguish the benefit of instance-adaptive selection from an improvement obtained by simply choosing a better global block size. It also exposes how fixed-block performance varies across backbones, tasks, and sampling temperatures.

Across the complete Qwen3 sweep, BlockPilot achieves the highest average speedup and acceptance length under both decoding temperatures. At temperature $= 0$, it reaches $4.17\times$ speedup with an average acceptance length of 6.59 on Qwen3-4B, and $4.66\times$ with 6.46 on Qwen3-8B. At temperature $= 1$, the corresponding results are $4.20\times$ with 5.92 and $3.94\times$ with 5.55, respectively. These comparisons include the additional fixed block sizes 13, 14, 15, 17, 18, and 19. For example, on Qwen3-4B at temperature $= 0$, BlockPilot improves over DFlash(16) from $3.99\times$ to $4.17\times$ in average speedup and from 6.31 to 6.59 in acceptance length. Under temperature $= 1$, the corresponding improvements are from $3.80\times$ to $4.20\times$ and from 5.35 to 5.92. The complete sweep further shows that increasing the fixed block size does not produce monotonic gains: performance rises near the training configuration but can decline when the block becomes excessively large.

The results on the instruction and code models show the same pattern. BlockPilot obtains average speedups of $3.25\times$ and $2.40\times$ on Llama-3.1-8B-Instruct at temperatures 0 and 1, respectively, and $4.12\times$ and $3.95\times$ on Qwen3-Coder-30B-A3B. Although stochastic sampling reduces the absolute speedup and acceptance length on several benchmarks, BlockPilot retains its advantage over the fixed-block alternatives. The improvement is also observed on both short-form reasoning benchmarks and longer or more open-ended generation tasks. The consistent gains across Math, Code, and Chat benchmarks indicate that instance-adaptive block selection generalizes across model families, task categories, and sampling settings.

H Relation to Feedback-Based Adaptive Draft-Length Control

Prior adaptive speculative-decoding methods address the trade-off between drafting more tokens per round and maintaining a high acceptance rate. They can be broadly viewed as reactive or predictive strategies. Reactive strategies, including feedback-style controllers used in DiffuSpec-like methods Li et al. (2025a), update a subsequent draft length from observations such as the accepted length, rejection position, or recent acceptance rate. Dynamic drafting methods such as EAGLE-2 Li et al. (2024) likewise adjust the speculative structure during decoding according to the quality of intermediate proposals. Prediction-based strategies instead estimate a viable speculative length from currently available model features, reducing some of the trial-and-error required by purely reactive adjustment. All these approaches share the objective of matching speculative parallelism to input difficulty, but they differ in when the decision is made, which signal drives it, and which generation unit is controlled.

BlockPilot specifically formulates DFlash block-size selection as an instance-level policy decision made before speculative decoding. A feedback controller can be abstracted as $b_{t+1} = g(b_t, s_t)$, where s_t is accumulated from previous draft–verify rounds. In contrast, BlockPilot predicts $\hat{B}(x)$ once from the target model's raw logits at the final prefill position and fixes the selected block size

for the subsequent generation. This formulation is supported by two empirical properties observed in this work: the measured sample-optimal DFlash block size varies across inputs but remains concentrated near the drafter's training block size, and the prefill logit vector is highly informative for distinguishing among these local choices. The novelty therefore does not lie in adaptivity alone. It lies in converting DFlash block-size selection into a supervised, instance-conditioned policy over a structured local action space, using a signal available before the first draft–verify round. Relative to feedback-based adjustment, the distinction is threefold: decision timing (pre-decoding rather than after verification feedback), information source (the prompt-conditioned prefill logits rather than past acceptance statistics), and optimization granularity (one sample-level DFlash policy rather than repeated step-level corrections).

Feedback control is conceptually compatible with DFlash, but a controller designed for conventional draft-length adjustment is not a drop-in solution. In autoregressive speculative decoding, changing the draft length primarily truncates or extends a sequential draft trajectory. In DFlash, the block size specifies the number of tokens proposed jointly through iterative denoising. Increasing it therefore changes both the parallel verification horizon and the distribution of the block-level proposal. The exact decomposition in Appendix A shows that a larger block adds possible accepted positions while potentially changing survival over positions shared with a smaller block; it does not imply monotonic acceptance or prove locality around the training size. The latter is established separately by the empirical sweep. Consequently, a low acceptance statistic alone does not identify whether the cause is an unsuitable block size, intrinsically difficult prompt content, or stochastic variation in a particular draft–verify round. Directly applying a controller whose update rule assumes a monotonic length–acceptance relationship may therefore move the block size in an unsuitable direction or oscillate among neighboring choices. Applying feedback control to DFlash would require redesigning its action space around the empirically supported local candidates and calibrating a DFlash-specific update rule and stability criterion. Thus, the distinction is not that feedback is inapplicable, but that its original control assumptions do not directly match block-diffusion drafting.

The timing of feedback also creates a cold-start distinction. No acceptance history is available for the first speculative round, so a reactive controller must begin from a fixed default or perform an exploratory update before it becomes sample-specific. This delay is important for short generations, in which only a few rounds may be available for the controller to reach an effective operating point. Moreover, acceptance length is a discrete and locally noisy observation; aggressive updates may alternate between neighboring block sizes, whereas conservative updates may respond too slowly. BlockPilot uses information already computed by target-model prefill and makes its sample-specific decision before any drafting begins. It applies the predictor once, introduces no recurrent controller state or per-step update, and leaves the DFlash draft and verification procedure unchanged. The selected policy is therefore available for the entire generation, including the first speculative step.

The one-shot design also has clear limitations. Because the selected block size remains fixed, BlockPilot cannot respond when generation difficulty changes substantially after the prompt, and an incorrect initial prediction is not corrected by later acceptance observations. Its supervised labels additionally require offline evaluation of the local candidate block sizes, and the predictor may need recalibration when the target model, drafter training configuration, or deployment distribution changes. By contrast, a feedback-based controller can react to within-sequence changes and may operate without sample-level oracle labels, although it must trade off adaptation speed, observation noise, and control stability. The two approaches are thus complementary rather than mutually exclusive: BlockPilot can provide an informed initial block size that removes the cold start, after which a feedback controller could make bounded local adjustments when sufficient verification evidence has accumulated. We leave the evaluation of such a hybrid policy to future work.

I Limitations and Future Work

Data Construction and Training. Our training set comprises 2,000 labeled samples, with each label obtained by evaluating the local candidate set $\{B-k,\ldots,B+k\}$. A speculative-decoding run with one block size typically takes approximately 3 seconds for a given model. With the default $k=2$, labeling a sample requires five runs, or approximately 15 seconds. Constructing the full dataset takes approximately 8.35 hours, and predictor training requires an additional 3.8 hours, for a total offline preparation time of approximately 12.15 hours. We consider this cost acceptable in

practice: it is incurred only once for a model, is excluded from the online generation path, and is amortized over all subsequent inference requests. Approximately twelve hours of offline preparation is modest relative to contemporary large-scale model training and fine-tuning, particularly given the recurring inference-speed improvements obtained after deployment. Candidate evaluations are also independent across samples and block sizes, allowing the construction process to be distributed across available devices and workers.

Future work can further reduce the offline cost through the following directions:

(1) **Early termination with candidate bounds.** During the evaluation of a block-size candidate, statistics from completed draft–verify rounds can be used to maintain a confidence interval for its final acceptance length. Evaluation can stop when the upper confidence bound falls below the lower confidence bound of the current best candidate. The interval can incorporate the observed mean and variance of accepted lengths together with the number of remaining generation rounds, and its stopping threshold can be selected to meet a target label-error rate. Ambiguous candidates would still run to completion, while clearly inferior choices could be discarded after only a fraction of the generation trajectory. Calibration on held-out samples would quantify the reduction in executed rounds and verify that early stopping rarely changes the selected block-size label.

(2) **Short-rollout surrogate evaluation.** Each candidate can first be evaluated for a small, fixed number of draft–verify rounds. Early acceptance rate, first-rejection position, prefix-survival probability, draft-token confidence, and the variation of these statistics across rounds can then be provided to a surrogate ranker. The ranker can be trained on fully enumerated samples to predict either the final acceptance length of each candidate or their relative ordering, after which full decoding is reserved for the most promising one or two block sizes. A confidence threshold would trigger exhaustive enumeration when the predicted ranking is uncertain. Periodically evaluating all candidates on a calibration subset would track distribution shift and determine when the surrogate requires updating.

(3) **Reuse across similar samples.** Samples can be grouped using their final-position prefill logits or a lower-dimensional projection learned from these logits. Fully enumerated representatives within each group provide an empirical distribution over suitable block sizes, which can initialize candidate rankings for nearby samples. Candidates that are consistently dominated within a sufficiently dense cluster can be omitted, while the remaining high-probability choices are evaluated normally. The reuse criterion can combine representation distance, cluster size, and label concentration so that weakly supported transfers do not affect data quality. Full enumeration remains available when a sample is far from existing representatives, falls near a cluster boundary, or belongs to a cluster with a diffuse block-size distribution.

(4) **Active data construction.** An initial predictor can be trained on a small, task-balanced seed set for which all block sizes are fully enumerated. The predictor is then applied to the remaining unlabeled pool, and additional full enumeration is concentrated on samples with high predictive entropy, a small margin between neighboring block-size scores, disagreement across independently trained predictors, or poor coverage in the current training set. Newly labeled samples are added in rounds, followed by predictor retraining and evaluation on a fixed validation set. Construction can stop when both classification accuracy and end-to-end speedup cease to improve across successive rounds. This procedure allocates the labeling budget to informative or underrepresented samples instead of exhaustively processing examples that the current policy already classifies confidently.

These directions reduce redundant full speculative-decoding runs while retaining exhaustive evaluation for uncertain samples, providing a path to lower offline cost without compromising label quality.

Table 11: Complete speedup ratios and average acceptance length τ on Qwen3 models across Math, Code, and Chat benchmarks. Q3-4B and Q3-8B denote Qwen3-4B and Qwen3-8B, respectively. DFlash(n) denotes DFlash with block size n.

Model	Method	Math						Code						Chat		Overall	
		GSM8K		MATH-500		AIME24		HumanEval		MBPP		SWE-Bench		MT-Bench		Avg.	
Temperature $= 0$		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Q3-4B	EAGLE-3	1.99×	3.30	1.83×	3.08	1.79×	3.05	1.84×	3.05	1.78×	2.95	1.39×	2.48	1.74×	3.02	1.70×	2.95
	DFlash (4)	2.29×	3.32	2.20×	3.48	2.12×	3.45	2.12×	3.22	2.16×	3.17	1.59×	2.58	1.45×	2.85	1.99×	3.15
	DFlash (8)	3.56×	5.16	3.59×	5.68	3.47×	5.64	3.19×	4.84	3.22×	4.71	1.95×	3.17	1.99×	3.91	2.99×	4.73
	DFlash (13)	4.25×	6.17	4.68×	7.41	4.41×	7.18	3.96×	6.02	3.95×	5.79	2.09×	3.39	2.46×	4.84	3.69×	5.83
	DFlash (14)	4.40×	6.38	4.97×	7.87	4.47×	7.27	4.10×	6.24	4.00×	5.86	2.12×	3.43	2.49×	4.91	3.79×	5.99
	DFlash (15)	4.59×	6.65	5.08×	8.05	4.62×	7.52	4.16×	6.32	4.20×	6.15	2.14×	3.47	2.54×	5.01	3.90×	6.17
	DFlash (16)	4.55×	6.59	5.25×	8.31	4.58×	7.45	4.28×	6.51	4.26×	6.24	2.33×	3.78	2.69×	5.29	3.99×	6.31
	DFlash (17)	4.66×	6.75	5.27×	8.35	4.71×	7.66	4.34×	6.59	4.25×	6.23	2.37×	3.84	2.71×	5.33	4.04×	6.39
	DFlash (18)	4.43×	6.42	5.17×	8.18	4.53×	7.37	4.14×	6.29	3.97×	5.82	2.26×	3.67	2.66×	5.24	3.88×	6.14
	DFlash (19)	4.09×	5.93	5.02×	7.95	4.34×	7.07	3.74×	5.69	3.77×	5.51	2.13×	3.45	2.41×	4.74	3.64×	5.76
	DFlash (32)	2.92×	4.24	3.22×	5.09	2.99×	4.86	2.56×	3.89	2.71×	3.97	1.69×	2.74	1.62×	3.19	2.53×	4.00
	BlockPilot	**4.76×**	**6.90**	**5.45×**	**8.62**	**4.80×**	**7.82**	**4.46×**	**6.77**	**4.38×**	**6.42**	**2.45×**	**3.98**	**2.85×**	**5.62**	**4.17×**	**6.59**
Q3-8B	EAGLE-3	1.94×	3.23	1.81×	3.02	1.79×	3.00	1.89×	3.17	1.69×	2.82	1.47×	2.45	1.63×	2.83	1.75×	2.93
	DFlash (4)	2.45×	3.29	2.50×	3.47	2.55×	3.44	2.45×	3.33	2.32×	3.21	1.96×	2.58	1.51×	2.58	2.25×	3.13
	DFlash (8)	3.76×	5.04	4.09×	5.67	4.22×	5.69	3.82×	5.19	3.51×	4.86	2.45×	3.23	2.01×	3.45	3.41×	4.73
	DFlash (13)	4.59×	6.15	5.22×	7.24	5.16×	6.96	4.63×	6.28	4.03×	5.57	2.67×	3.51	2.63×	4.51	4.13×	5.75
	DFlash (14)	4.68×	6.28	5.34×	7.40	5.31×	7.16	4.74×	6.43	4.10×	5.67	2.68×	3.53	2.65×	4.53	4.21×	5.86
	DFlash (15)	4.82×	6.47	5.50×	7.62	5.43×	7.33	4.80×	6.51	4.22×	5.84	2.72×	3.58	2.74×	4.69	4.32×	6.01
	DFlash (16)	4.89×	6.56	5.98×	8.29	5.57×	7.51	4.98×	6.76	4.33×	5.99	2.72×	3.58	2.48×	4.25	4.42×	6.13
	DFlash (17)	4.99×	6.69	5.73×	7.95	5.54×	7.47	4.99×	6.78	4.15×	5.74	2.68×	3.53	2.81×	4.82	4.41×	6.14
	DFlash (18)	4.97×	6.66	5.68×	7.88	5.49×	7.40	4.91×	6.67	4.13×	5.71	2.63×	3.47	2.71×	4.65	4.36×	6.06
	DFlash (19)	4.94×	6.63	5.64×	7.82	5.46×	7.36	4.84×	6.57	4.10×	5.68	2.62×	3.44	2.64×	4.53	4.32×	6.00
	DFlash (32)	4.47×	5.99	5.66×	7.84	5.20×	7.01	4.66×	6.32	3.92×	5.42	2.50×	3.29	2.37×	4.06	4.11×	5.70
	BlockPilot	**5.02×**	**6.73**	**6.17×**	**8.56**	**6.16×**	**8.30**	**5.21×**	**7.07**	**4.53×**	**6.26**	**2.96×**	**3.90**	**2.55×**	**4.37**	**4.66×**	**6.46**
Temperature $= 1$		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Q3-4B	EAGLE-3	1.89×	3.22	1.75×	2.99	1.64×	2.79	1.74×	3.01	1.69×	2.89	1.47×	2.31	1.77×	2.99	1.70×	2.88
	DFlash (4)	2.32×	3.17	2.34×	3.26	2.08×	2.90	2.29×	3.11	2.25×	3.10	1.79×	2.36	1.62×	2.74	2.10×	2.95
	DFlash (8)	3.60×	4.91	3.71×	5.16	2.97×	4.13	3.37×	4.58	3.21×	4.42	2.17×	2.86	2.24×	3.79	3.04×	4.26
	DFlash (13)	4.07×	5.55	4.49×	6.24	3.47×	4.83	3.95×	5.38	3.99×	5.49	2.27×	2.99	2.82×	4.77	3.58×	5.04
	DFlash (14)	3.97×	5.42	4.81×	6.69	3.46×	4.82	4.08×	5.55	3.87×	5.34	2.22×	2.92	2.98×	5.03	3.63×	5.11
	DFlash (15)	4.17×	5.69	4.72×	6.57	3.64×	5.06	3.98×	5.42	4.01×	5.53	2.47×	3.25	2.79×	4.71	3.68×	5.18
	DFlash (16)	4.36×	5.95	4.83×	6.72	3.66×	5.10	4.32×	5.87	4.02×	5.54	2.33×	3.07	3.10×	5.23	3.80×	5.35
	DFlash (17)	4.49×	6.13	4.96×	6.90	3.69×	5.13	4.28×	5.82	4.28×	5.89	2.24×	2.94	3.05×	5.15	3.85×	5.43
	DFlash (18)	4.28×	5.84	4.72×	6.57	3.54×	4.92	4.23×	5.75	3.85×	5.30	2.19×	2.88	2.84×	4.80	3.66×	5.15
	DFlash (19)	3.82×	5.21	4.90×	6.82	3.34×	4.66	4.07×	5.54	3.69×	5.09	2.10×	2.77	2.66×	4.49	3.51×	4.94
	DFlash (32)	2.83×	3.87	3.35×	4.66	2.79×	3.88	2.76×	3.76	2.69×	3.71	1.91×	2.52	1.91×	3.22	2.61×	3.66
	BlockPilot	**4.77×**	**6.51**	**5.41×**	**7.53**	**4.02×**	**5.60**	**4.74×**	**6.45**	**4.59×**	**6.33**	**2.58×**	**3.39**	**3.31×**	**5.59**	**4.20×**	**5.92**
Q3-8B	EAGLE-3	1.87×	3.12	1.73×	2.91	1.63×	2.74	1.75×	3.05	1.64×	2.74	1.33×	2.11	1.58×	2.70	1.65×	2.77
	DFlash (4)	2.34×	3.17	2.25×	3.22	2.05×	2.92	2.25×	3.05	2.23×	3.03	1.66×	2.20	1.44×	2.45	2.03×	2.86
	DFlash (8)	3.48×	4.72	3.53×	5.06	2.92×	4.17	3.26×	4.41	3.20×	4.36	1.94×	2.57	1.93×	3.28	2.89×	4.08
	DFlash (13)	4.00×	5.43	4.39×	6.29	3.41×	4.86	3.90×	5.28	3.73×	5.07	2.10×	2.79	2.41×	4.10	3.42×	4.83
	DFlash (14)	4.04×	5.48	4.42×	6.33	3.42×	4.87	3.91×	5.30	3.79×	5.15	2.01×	2.67	2.35×	4.00	3.42×	4.83
	DFlash (15)	4.03×	5.47	4.79×	6.86	3.47×	4.95	3.90×	5.28	3.75×	5.10	2.20×	2.92	2.34×	3.99	3.50×	4.94
	DFlash (16)	4.28×	5.81	4.62×	6.62	3.49×	4.98	4.19×	5.68	3.98×	5.41	2.09×	2.77	2.20×	3.75	3.55×	5.00
	DFlash (17)	4.39×	5.95	4.56×	6.53	3.55×	5.07	4.20×	5.69	3.99×	5.43	2.15×	2.85	2.58×	4.39	3.63×	5.13
	DFlash (18)	4.13×	5.60	4.45×	6.38	3.52×	5.01	4.10×	5.55	3.75×	5.10	2.08×	2.76	2.54×	4.33	3.51×	4.96
	DFlash (19)	4.16×	5.64	4.68×	6.70	3.53×	5.04	4.25×	5.76	3.72×	5.06	2.04×	2.71	2.50×	4.25	3.55×	5.02
	DFlash (32)	4.10×	5.57	4.41×	6.32	3.39×	4.84	3.77×	5.11	3.68×	5.00	2.01×	2.67	2.23×	3.79	3.37×	4.75
	BlockPilot	**4.69×**	**6.37**	**5.06×**	**7.25**	**3.84×**	**5.47**	**4.65×**	**6.30**	**4.54×**	**6.17**	**2.30×**	**3.05**	**2.48×**	**4.23**	**3.94×**	**5.55**

Table 12: Speedup ratios and average acceptance length τ on Llama and Qwen models across Math, Code, and Chat benchmarks. Llama denotes Llama-3.1-8B-Instruct, and Qwen denotes Qwen3-Coder-30B-A3B. DFlash(n) denotes DFlash with block size n.

Model	Method	Math						Code						Chat		Overall	
		GSM8K		MATH-500		AIME24		HumanEval		MBPP		SWE-Bench		MT-Bench		Avg.	
Temperature $= 0$		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Llama	EAGLE-3	1.81×	3.06	1.70×	2.98	1.64×	2.99	1.94×	3.25	1.87×	3.22	1.60×	2.80	1.50×	2.72	1.72×	3.00
	DFlash (4)	2.21×	3.16	2.07×	3.08	2.00×	3.09	2.36×	3.35	2.28×	3.32	1.95×	2.90	1.83×	2.82	2.10×	3.10
	DFlash (8)	2.89×	4.13	2.93×	4.37	3.03×	4.69	3.42×	4.86	3.39×	4.94	2.51×	3.74	2.33×	3.59	2.93×	4.33
	DFlash (13)	2.93×	4.18	3.18×	4.74	3.29×	5.10	3.71×	5.26	3.71×	5.40	2.61×	3.90	2.52×	3.87	3.14×	4.64
	DFlash (14)	2.90×	4.13	3.17×	4.72	3.25×	5.03	3.69×	5.23	3.72×	5.41	2.60×	3.88	2.47×	3.80	3.11×	4.60
	DFlash (15)	2.89×	4.12	3.16×	4.71	3.22×	4.99	3.65×	5.18	3.67×	5.34	2.60×	3.87	2.47×	3.80	3.09×	4.57
	DFlash (16)	2.91×	4.16	3.11×	4.63	2.93×	4.54	3.67×	5.21	3.57×	5.20	2.52×	3.76	2.52×	3.88	3.03×	4.48
	DFlash (17)	2.88×	4.12	2.83×	4.22	2.85×	4.41	3.67×	5.20	3.66×	5.33	2.58×	3.85	2.70×	4.16	3.02×	4.47
	DFlash (18)	2.88×	4.11	2.82×	4.21	2.84×	4.40	3.64×	5.17	3.67×	5.34	2.58×	3.85	2.71×	4.17	3.02×	4.46
	DFlash (19)	2.89×	4.13	2.83×	4.21	2.84×	4.40	3.60×	5.11	3.65×	5.32	2.56×	3.82	2.69×	4.14	3.01×	4.45
	DFlash (32)	2.87×	4.09	2.79×	4.16	2.80×	4.33	3.59×	5.09	3.57×	5.20	2.51×	3.74	2.66×	4.09	2.97×	4.39
	BlockPilot	**2.96×**	**4.22**	**3.21×**	**4.79**	**3.63×**	**5.62**	**3.74×**	**5.31**	**3.75×**	**5.46**	**2.71×**	**4.04**	**2.78×**	**4.28**	**3.25×**	**4.82**
Qwen	EAGLE-3	1.75×	2.94	1.85×	3.07	1.80×	3.00	1.92×	3.41	1.89×	3.31	1.47×	2.53	1.21×	2.34	1.70×	2.94
	DFlash (4)	2.13×	3.04	2.26×	3.17	2.19×	3.10	2.34×	3.51	2.30×	3.41	1.79×	2.63	1.48×	2.44	2.07×	3.04
	DFlash (8)	2.94×	4.20	3.34×	4.70	3.04×	4.32	3.86×	5.79	3.69×	5.47	2.21×	3.24	2.01×	3.32	3.01×	4.43
	DFlash (13)	3.42×	4.88	4.06×	5.71	3.47×	4.92	5.18×	7.77	4.71×	6.98	2.44×	3.59	2.23×	3.67	3.64×	5.36
	DFlash (14)	3.51×	5.02	4.11×	5.77	3.69×	5.24	5.39×	8.07	4.85×	7.18	2.33×	3.43	2.33×	3.85	3.74×	5.51
	DFlash (15)	3.55×	5.07	4.07×	5.73	3.76×	5.34	5.63×	8.44	4.94×	7.32	2.49×	3.66	2.33×	3.85	3.82×	5.63
	DFlash (16)	3.58×	5.12	4.16×	5.85	3.67×	5.20	5.74×	8.61	5.11×	7.58	2.48×	3.65	2.29×	3.77	3.86×	5.68
	DFlash (17)	3.60×	5.14	4.19×	5.89	3.98×	5.65	5.96×	8.93	4.98×	7.39	2.58×	3.79	2.31×	3.81	3.94×	5.80
	DFlash (18)	3.49×	4.99	4.15×	5.84	3.59×	5.09	5.86×	8.79	5.05×	7.49	2.37×	3.48	2.34×	3.86	3.84×	5.65
	DFlash (19)	3.45×	4.93	4.13×	5.81	3.66×	5.20	5.93×	8.89	5.20×	7.72	2.38×	3.50	2.31×	3.82	3.87×	5.70
	DFlash (32)	3.38×	4.83	3.91×	5.49	3.43×	4.86	5.60×	8.40	4.87×	7.22	2.35×	3.46	2.21×	3.64	3.68×	5.42
	BlockPilot	**3.75×**	**5.36**	**4.37×**	**6.14**	**4.19×**	**5.95**	**6.05×**	**9.07**	**5.26×**	**7.80**	**2.70×**	**3.96**	**2.49×**	**4.10**	**4.12×**	**6.05**
Temperature $= 1$		*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ	*Speedup*	τ
Llama	EAGLE-3	1.51×	2.79	1.19×	2.28	0.76×	1.35	1.74×	2.95	1.72×	2.94	1.24×	2.20	0.92×	1.76	1.30×	2.32
	DFlash (4)	1.84×	2.89	1.45×	2.38	0.93×	1.45	2.12×	3.05	2.10×	3.04	1.51×	2.30	1.12×	1.86	1.58×	2.42
	DFlash (8)	2.16×	3.40	1.67×	2.75	1.11×	1.74	2.94×	4.22	2.92×	4.24	1.47×	2.24	1.42×	2.36	1.96×	2.99
	DFlash (13)	2.10×	3.30	1.81×	2.97	1.12×	1.75	3.15×	4.53	3.35×	4.86	1.65×	2.52	1.63×	2.70	2.11×	3.23
	DFlash (14)	2.19×	3.45	1.68×	2.76	1.07×	1.67	2.97×	4.26	2.69×	3.90	1.49×	2.28	1.30×	2.16	1.91×	2.92
	DFlash (15)	2.21×	3.48	1.80×	2.96	0.93×	1.46	3.16×	4.54	2.96×	4.29	1.77×	2.70	1.67×	2.78	2.07×	3.17
	DFlash (16)	2.45×	3.85	1.75×	2.88	1.15×	1.80	3.00×	4.31	2.80×	4.06	1.60×	2.44	1.54×	2.56	2.04×	3.13
	DFlash (17)	2.17×	3.41	1.79×	2.94	1.21×	1.90	2.92×	4.20	3.15×	4.57	1.40×	2.14	1.52×	2.53	2.02×	3.10
	DFlash (18)	2.54×	3.99	1.72×	2.84	1.24×	1.94	2.70×	3.88	3.05×	4.42	1.71×	2.61	1.71×	2.84	2.10×	3.22
	DFlash (19)	2.27×	3.57	1.88×	3.09	1.05×	1.65	3.03×	4.35	2.87×	4.17	1.71×	2.61	1.48×	2.46	2.04×	3.13
	DFlash (32)	1.76×	2.76	1.65×	2.71	0.99×	1.55	2.85×	4.10	3.05×	4.42	1.52×	2.32	1.54×	2.56	1.91×	2.92
	BlockPilot	**2.73×**	**4.29**	**2.10×**	**3.45**	**1.48×**	**2.32**	**3.29×**	**4.72**	**3.37×**	**4.88**	**1.89×**	**2.88**	**1.92×**	**3.20**	**2.40×**	**3.68**
Qwen	EAGLE-3	1.69×	2.83	1.71×	2.96	1.59×	2.74	1.91×	3.34	1.82×	3.20	1.34×	2.27	1.20×	2.27	1.61×	2.80
	DFlash (4)	2.06×	2.93	2.09×	3.06	1.94×	2.84	2.33×	3.44	2.22×	3.30	1.63×	2.37	1.46×	2.37	1.96×	2.90
	DFlash (8)	2.89×	4.10	3.05×	4.47	2.56×	3.76	3.77×	5.58	3.67×	5.44	1.96×	2.86	2.01×	3.28	2.84×	4.21
	DFlash (13)	3.37×	4.79	3.63×	5.32	3.01×	4.42	5.20×	7.68	4.45×	6.60	2.06×	3.01	2.15×	3.51	3.41×	5.05
	DFlash (14)	3.41×	4.85	3.66×	5.37	3.10×	4.55	5.68×	8.40	4.76×	7.06	2.08×	3.03	2.30×	3.75	3.57×	5.29
	DFlash (15)	3.58×	5.08	3.58×	5.25	3.06×	4.50	5.85×	8.66	4.71×	6.99	2.16×	3.15	2.30×	3.74	3.61×	5.34
	DFlash (16)	3.41×	4.84	3.72×	5.45	3.09×	4.54	5.27×	7.79	4.91×	7.28	2.09×	3.04	2.27×	3.70	3.54×	5.23
	DFlash (17)	3.59×	5.10	3.64×	5.34	3.06×	4.49	5.99×	8.86	4.98×	7.39	2.21×	3.23	2.41×	3.92	3.70×	5.48
	DFlash (18)	3.44×	4.89	3.64×	5.33	3.02×	4.44	5.66×	8.37	4.87×	7.23	2.14×	3.11	2.31×	3.77	3.58×	5.30
	DFlash (19)	3.41×	4.85	3.74×	5.49	2.97×	4.36	5.71×	8.44	4.81×	7.14	2.13×	3.11	2.32×	3.77	3.58×	5.31
	DFlash (32)	3.41×	4.84	3.61×	5.29	2.88×	4.22	5.54×	8.19	4.89×	7.25	2.09×	3.05	2.08×	3.38	3.50×	5.17
	BlockPilot	**3.77×**	**5.36**	**4.01×**	**5.87**	**3.33×**	**4.89**	**6.35×**	**9.39**	**5.34×**	**7.92**	**2.36×**	**3.44**	**2.48×**	**4.04**	**3.95×**	**5.84**