SpecBlock: Block-Iterative Speculative Decoding with Dynamic Tree Drafting

Weijie Shi[1*] **Qiang Xu**[2] **Fan Deng**[2] **Yaguang Wu**[2] **Jiarun Liu**[2]
Yehong Xu[1] **Hao Chen**[1] **Jia Zhu**[3] **Jiajie Xu**[4] **Xiangjun Huang**[2†]
Jian Yang[2] **Xiaofang Zhou**[1,5†]

[1]Hong Kong University of Science and Technology [2]MetaX
[3]Zhejiang Normal University [4]Soochow University
[5]GuangZhou HKUST Fok Ying Tung Research Institute

Abstract

Speculative decoding accelerates LLM inference by drafting a tree of candidate continuations and verifying it in one target forward. Existing drafters fall into two camps with opposite weaknesses. Autoregressive drafters such as EAGLE-3 preserve dependence along each draft path but call the drafter once per tree depth, making drafting a non-trivial share of per-iteration latency. Parallel drafters cut drafter calls by predicting multiple future positions in one forward, but each position is predicted without seeing the others, producing paths the verifier rejects. In this paper, we propose SpecBlock, a block-iterative drafter that combines path dependence with cheap drafting. Each drafter forward produces K dependent positions and we call this a block. The draft tree grows through repeated block expansions. Two mechanisms explicitly carry path dependence to keep later draft positions accurate. Within each block, a layer-wise shift carries the previous position's hidden state into every decoder layer. Across blocks, each new block can start from any position of the previous block, inheriting its hidden state to extend the path. To spend verifier budget where acceptance is likely, a co-trained rank head replaces the fixed top-k tree by allocating per-position branching during drafting. To avoid training the drafter on prefixes it never produces at inference, a valid-prefix mask drops the loss at later positions once an earlier one is wrong. Beyond static drafting, a cost-aware bandit at deployment uses free verifier feedback to update the drafter selectively, only when the expected throughput gain exceeds the update cost. Experiments show that SpecBlock improves mean speedup by 8–13% over EAGLE-3 at 44–52% of its drafting cost, and cost-aware adaptation extends this lead to 11–19%. On MetaX C500, SpecBlock also improves mean throughput over EAGLE-3 by 14–16% under greedy decoding.

1 Introduction

Since large language model (LLM) decoding is often limited by memory bandwidth, speculative decoding [1, 2, 3, 4, 5] addresses this bottleneck by using a small draft model to predict multiple future tokens and letting the target model verify these candidates in parallel, allowing one target forward to accept multiple tokens and use the available compute capacity more fully. Tree-based verification [3, 6] further replaces a single drafted sequence with a draft tree, giving the verifier multiple alternatives at future positions and substantially increasing the acceptance length. The gain from tree-based verification depends on a balance: the tree must cover continuations the target is

*Email: shiweijie0311@foxmail.com.
†Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

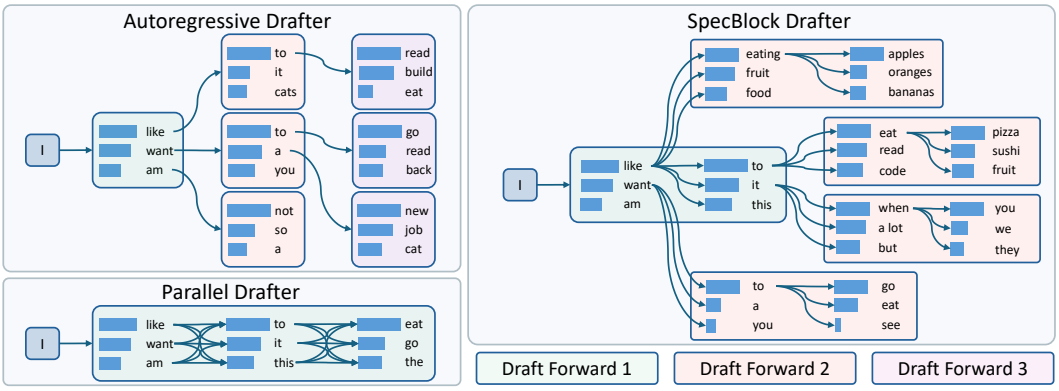


Figure 1: Three drafting paradigms. Autoregressive drafters (**left**) add one depth per drafter call. Parallel drafters (**bottom-left**) predict all depths independently in one call. SpecBlock (**right**) produces K dependent positions per call and batches blocks from multiple starting positions into each subsequent call, growing the tree iteratively.

likely to accept while keeping draft computation small enough that the saved target calls translate into net speedup.

Autoregressive drafters such as EAGLE-3 [7, 8, 9] grow the draft tree depth by depth. This preserves dependence along each draft path and can reach average acceptance lengths near 6, but each added tree depth still costs one more sequential drafter round. Although the drafter is small, each round is itself memory-bound, so the serial calls accumulate bursts of weight loading and consume close to 30% of per-iteration latency on 8B-level target. Parallel drafters [10] reduce this overhead by proposing several future positions in one call, shrinking drafting to roughly 7%. However, once alternatives from different depths are combined into a draft tree, they form a large combinatorial space in which many paths are not coherent continuations, and the verifier wastes budget on them. This calls for a balance between the two camps, a drafter that both makes few drafter calls and preserves path coherence along each draft path, as illustrated in Figure 1.

To realize this balance, we propose **SpecBlock**, a block-iterative drafter that treats each draft forward as producing a multi-token block and grows the draft tree through repeated block expansions. A generated tree node can serve as the starting point of a subsequent block, so one batched draft forward can extend multiple branches in parallel rather than expanding the tree depth by depth. Two mechanisms keep later positions accurate inside this construction by explicitly carrying dependence. Within each block, a layer-wise shift carries the previous position's hidden state into every decoder layer. Across blocks, each new block can continue from any position of the previous block, conditioned on that position's hidden state.

Rank-guided tree construction. Different draft positions deserve different amounts of branching, because the target token may sit at the top of the draft distribution at one position and far down at another. A co-trained rank head reads each position's hidden state and predicts how high the target token ranks in that position's draft distribution, expressed as a coarse bucket. This bucket sets the number of sibling alternatives at the position and decides whether the position starts a later block, so the tree is shaped on the fly during drafting rather than pruned afterwards.

Valid-prefix curriculum learning. An autoregressive drafter teacher-forces each step from a fresh ground-truth prefix, so every step's loss is supervised under the correct context. SpecBlock cannot do this, because its K predictions are produced jointly in one forward and later positions read the actual earlier predictions instead of the ground-truth prefix. If an earlier prediction is wrong, the verifier rejects the entire path. Supervising later positions on the ground-truth target then only spends capacity on tokens the drafter will never commit. The valid-prefix mask therefore drops the loss at any later position once an earlier one on the same path is wrong.

Cost-aware serving-time adaptation. A small drafter trained offline cannot fit every domain, and acceptance length drops when the serving prompt distribution moves away from the training mix. The verifier already produces a free adaptation signal at every query, namely the target distribution at each rejected position, which is computed during verification at no extra cost. A cost-aware bandit

reads this signal and decides whether to skip the update, update only the output heads, or update the full drafter, taking a non-skip action only when the expected throughput gain exceeds the update cost.

Experiments show that SpecBlock improves mean speedup by 8–13% over EAGLE-3 at 44–52% of its drafting cost. Cost-aware serving-time adaptation widens this advantage to 11–19% on benchmarks with sufficient streaming queries. Additional experiments on MetaX C500 in Appendix J show 14–16% higher mean throughput than EAGLE-3 under greedy decoding. The code is available at https://github.com/shiweijiezero/SpecBlock.

2 Related Work

Autoregressive drafters. Standard speculative decoding [1, 2] establishes a lossless draft-then-verify framework, where a small drafter proposes a chain of future tokens that the target verifies in parallel. SpecInfer [3] generalizes the chain into a token tree so one verification accepts the longest matching path among multiple candidates, lifting accepted length when the drafter is uncertain. The EAGLE family [7, 8, 9] then trades drafter capacity for fidelity by autoregressing in the target model's feature space and growing dynamic trees from drafter confidence, with HASS [11] further aligning training and inference by simulating the drafter's own multi-step rollout. Other variants cut drafter overhead through distillation [5], by reusing the target's own shallow layers [12, 13], or by retrieving cached continuations from a datastore [14, 15]. All of these methods gain acceptance by propagating dependence one depth at a time, so each added depth still costs another drafter step.

Parallel and blockwise drafters. Parallel and blockwise drafters take the opposite trade-off and cut the drafter to one forward by predicting several future positions at once. Each position is predicted independently of the others, so the draft tree fails to capture the dependence between adjacent tokens, and its paths diverge from the target's continuation after the first few depths. Draft-head methods [16, 10] attach independent heads at fixed offsets. Mask-based drafters predict each future offset from a learnable mask token, with BiTA [17], ParallelSpec [18], and PARD [19] differing in how the masks are integrated, and DART [20] layering a diffusion-style masked-prediction objective on top. Hydra [21] re-introduces dependence by chaining the heads sequentially, each conditioning on the candidate continuation produced by earlier heads. Blockwise and semi-autoregressive variants instead enlarge the draft unit, through layer cascades [22], semi-autoregressive block drafting [23, 24], or recurrent and block-mask architectures [25, 26, 27], raising the average tokens per draft call. Falcon is the closest of these and also drafts semi-autoregressive blocks, but carries within-block dependence through stacked LSTM layers and relaxed-causal-mask attention that lets all positions inside a block see one another, and verifies a hand-crafted static decoding tree. SpecBlock instead enforces strict left-to-right within-block dependence through a per-layer hidden-state shift, and shapes the verifier tree dynamically through a co-trained rank head.

Tree construction. On top of an autoregressive drafter, the draft tree is shaped externally from drafter signals to decide which nodes the target verifies. Sequoia [6] solves an offline DP over tree size and depth, while C2T [28], OPT-Tree [29], DySpec [30], and TALON [31] adapt the tree from drafter probability, confidence, or budget signals. SpecBlock instead integrates tree construction into the drafter through a rank head that sets per-position branching and which positions start later blocks.

Serving-time adaptation. Speculative drafters are kept small to make drafting cheap, which also makes them sensitive to serving-time distribution shifts that lower acceptance length. One line leaves drafter weights frozen and adapts only the speculation hyperparameters such as proposal length and tree size, either via a bandit over candidate configurations [32] or via a learned threshold on per-position acceptance probability [33]. Another updates the drafter through verifier-feedback distillation [34], with extensions reformulating the drafter as a self-speculative head trained with a KL-to-RL schedule on accepted-position rewards [35], scheduling updates over time [36], or integrating training tighter into the serving stack [37], but each commits the trainable parameter subset ahead of time. Beyond when to update, SpecBlock makes which subset of drafter parameters to refresh a per-query decision, exposing a heads-versus-full-drafter action split where the rank head and `lm_head` form a self-contained output-side pathway that can absorb output-level mistakes without touching the decoder.

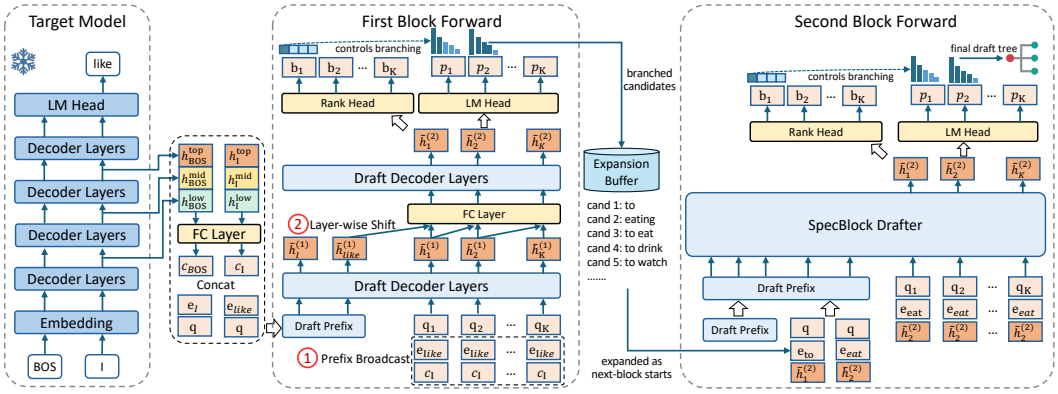


Figure 2: SpecBlock drafter architecture and block-iterative drafting. The first block (middle) fuses the target's multi-layer hidden state, the verified token, and K position queries; prefix broadcast (①) shares the target features across the K positions, and a layer-wise shift (②) propagates position $k-1$'s state into position k between consecutive decoder layers. The lm head outputs draft distributions p_k and the rank head outputs bucket labels b_k that control per-position branching width. Branched candidates fill the expansion buffer and feed block 2 (right) as next-block starts.

3 SpecBlock

Let $\mathcal{M}$ be the target model and $\mathcal{D}_\theta$ a small drafter that proposes a tree of candidate continuations for $\mathcal{M}$ to verify in one parallel forward. Speculative decoding throughput is the ratio $\Phi = \tau \,/\, (T_{\mathcal{M}} + T_{\mathcal{D}})$, where τ is the average length accepted per verifier call, $T_{\mathcal{M}}$ is the time of one target forward, and $T_{\mathcal{D}}$ is the cost of all drafter calls used to assemble that tree. Autoregressive drafters keep τ high but invoke the drafter once per tree depth, paying $T_{\mathcal{D}}$ proportional to depth. Parallel drafters collapse $T_{\mathcal{D}}$ to a single forward but lose τ because each future position is predicted without seeing the others.

SpecBlock improves Φ by sitting between these extremes: each drafter forward produces K dependent positions, and the tree past depth K is grown by re-using the drafter on a batch of starting points selected from earlier blocks. We further shape the tree's branching during drafting through a co-trained rank head, train the drafter under the prefix distribution it actually faces at inference, and refresh it selectively at serving time with a verifier-derived update signal.

3.1 The drafter block

Like prior speculative sampling methods, SpecBlock alternates between drafting and verification. The difference from EAGLE-3 [9] lies in the drafting stage, where each drafter forward predicts K consecutive positions in parallel as one block.

Block forward. Consider a draft block at the verified prefix's last position t, illustrated in Figure 2. Following EAGLE-3, we build a context feature c_t by concatenating the target model $\mathcal{M}$'s low-, mid-, and top-layer hidden states at position t and projecting them to the drafter's dimension d via a learned linear projection W_{cond}:

$$c_t \;=\; W_{\text{cond}}\,[\,h_t^{\text{low}}\,,\; h_t^{\text{mid}}\,,\; h_t^{\text{top}}\,]. \tag{1}$$

The other two inputs to the drafter are the embedding of the last committed token x_t and K learnable position queries $\mathbf{q}_1, \ldots, \mathbf{q}_K$, one per draft depth. The three signals are normalized and fused into a per-position input via a learned linear projection W_{fuse}:

$$h_{t,k}^{(0)} \;=\; W_{\text{fuse}}\big[\,\text{norm}(c_t)\,,\; \text{norm}(\text{embed}(x_t))\,,\; \text{norm}(\mathbf{q}_k)\,\big]. \tag{2}$$

A prefix broadcast ties the same c_t and $\text{embed}(x_t)$ across positions so each one receives the prefix context directly. Only $\mathbf{q}_k$ varies across positions. This per-position input then passes jointly through the drafter's L Transformer decoder layers, which match $\mathcal{M}$'s per-layer architecture, giving the last-layer state $h_{t,k}^{(L)}$ at each position. The lm_head reads $h_{t,k}^{(L)}$ to produce the draft distribution $p_{t,k}$, and we cache $h_{t,k}^{(L)}$ for downstream blocks.

Within-block dependence. The K positions are produced jointly. Any coherence along a draft path must therefore come from interactions inside the L decoder layers. Cross-position causal attention restricts position k to attend only to positions $\leq k$ within the block, plus preceding blocks via cached key-value pairs, reproducing left-to-right dependence at the attention level. However, each attended position contributes only one weight in the softmax mixture, and that weight is diluted as the prefix grows, collapsing acceptance at deeper positions of a block. We therefore add a layer-wise shift between consecutive decoder layers that explicitly carries position $k{-}1$'s state into position k. This approximates in one forward the state propagation that EAGLE-3 obtains by running a separate drafter forward per position. Before entering layer $\ell{+}1$, position k's state is concatenated with position $k{-}1$'s state from the same layer and projected back to $\mathbb{R}^d$ via a per-layer learned linear projection $W^{(\ell)}_{\text{shift}}$,

$$\tilde{h}^{(\ell)}_{t,k} \;=\; W^{(\ell)}_{\text{shift}}\big[\, h^{(\ell)}_{t,k}\,,\; h^{(\ell)}_{t,k-1}\,\big], \tag{3}$$

with the convention $h^{(\ell)}_{t,0} = h^{(\ell)}_{t,1}$ at $k{=}1$. This recovers the dependence lost to attention dilution while staying within a single drafter forward.

3.2 Rank-guided tree expansion

A draft tree grows along two axes: depth, by chaining additional drafter forwards past the first block, and width, by attaching sibling alternatives at each position. The verifier budget along both axes should track the drafter's uncertainty. At an easy position one child suffices, while at a harder position the target sits several ranks deeper and the path is recovered only if at least one of several alternatives matches. A fixed branching factor either over-spends on easy positions or under-explores hard ones. A co-trained rank head coordinates both axes. Its bucket prediction at each position determines both the per-position branching width and whether the position starts a later block.

Rank head. The rank prediction needs features that reflect both the drafter's internal confidence and the shape of its output distribution. The rank head g_ϕ reads two such features at each position: the last-layer hidden state $h^{(L)}_{t,k} \in \mathbb{R}^d$, which carries the drafter's contextual representation, and a fixed 15-dimensional summary $\psi(p_{t,k})$ of the draft distribution $p_{t,k}$, detailed in Appendix B. Both inputs are detached from the drafter's gradient via the stop-gradient operator $\mathrm{sg}(\cdot)$,

$$g_\phi\big(\big[\,\mathrm{sg}(h^{(L)}_{t,k})\,,\; \mathrm{sg}(\psi(p_{t,k}))\,\big]\big) \;\in\; \{b_0, b_1, b_2, b_3\}, \tag{4}$$

so that the rank objective shapes the head's parameters but not the drafter trunk, leaving token prediction unaffected.

Bucket-driven branching. The optimal branching factor changes sharply with rank, not smoothly. Per-rank training samples are also highly imbalanced, with rank-1 dominating and distant ranks rare. We therefore collapse the rank prediction into four coarse buckets and assign each bucket a branching factor b, so position k attaches the top-b tokens of $p_{t,k}$ as siblings within its block. Confident positions attach few siblings since the target is already near the top of the drafter's distribution, while uncertain positions attach more siblings to widen the recovery window.

Cross-block iteration. Cross-block iteration re-invokes the drafter from positions whose rank-head bucket schedules them as next-block starts. These positions are batched into one drafter forward to produce K further positions from each. The condition c_t at each such point is no longer the target model's hidden state but the drafter's own cached $h^{(L)}_{t,k}$, which is already in $\mathbb{R}^d$ and bypasses W_{cond}. We use the drafter's self-produced features here because the target has not yet verified the position, so no target hidden state is available. We bound the chain at M blocks, so the longest path in the tree reaches depth $M{\cdot}K$ at the cost of M drafter forwards.

3.3 Valid-prefix curriculum learning

The drafter and rank head should be trained under conditions consistent with inference. An autoregressive drafter teacher-forces each step on the ground-truth prefix, so every supervision signal sees a right-prefix context. SpecBlock cannot do the same. All K positions of a block are produced jointly in one forward, with position k's hidden state built from the drafter's own representations at

earlier positions, so ground-truth tokens cannot be spliced in mid-forward. When an earlier prediction is wrong, later positions are supervised under a wrong-prefix context, which both interferes with right-prefix supervision and is wasted because the verifier truncates the path at the first deviation. We therefore mask both the draft loss and the rank-head loss on any path within the block that has deviated.

Valid-prefix mask. We define a binary mask $m_{t,k} \in \{0, 1\}$ along each path of a block. The mask is initialized to $m_{t,1} = 1$ at the first position of every path. After each draft position, the mask updates by

$$m_{t,k+1} \;=\; m_{t,k} \cdot \mathbb{1}\big[\arg\max p_{t,k} \;=\; y^{\star}_{t,k}\big]\,, \tag{5}$$

where $y^{\star}_{t,k}$ is the target token at the offset that draft position k predicts. We compute position k's draft loss only on the paths the mask still admits,

$$\mathcal{L}_k^{\text{draft}} \;=\; -\frac{1}{N_k}\sum_t m_{t,k}\sum_v p^{\star}_{t,k}(v)\,\log p_{t,k}(v), \qquad N_k \;=\; \sum_t m_{t,k}, \tag{6}$$

where $p^{\star}_{t,k}$ is the target's next-token distribution at the matching offset.

Rank-head supervision. For each training position we compute the target token's rank r within $p_{t,k}$ and assign the bucket label by the rule $r{=}1 \mapsto b_0$, $r \in [2, 4] \mapsto b_1$, $r \in [5, 10] \mapsto b_2$, $r > 10 \mapsto b_3$. The rank head is supervised with cross-entropy against this label, masked by the same valid-prefix mask $m_{t,k}$.

Cross-block training. Inference past the first block conditions on the drafter's own cached hidden state rather than on the target's multi-layer feature. To expose the drafter to this shift during training, at each block boundary we sample a cut position s uniformly from $\{1, \ldots, K\}$, take the current block's last-layer hidden state $h^{(L)}_{t,s}$ as the next block's condition, and shift the ground-truth token sequence by s positions as the next block's input. Uniform sampling of s covers the full range of cross-block splits the rank head can produce at inference.

Total objective. The drafter is trained end-to-end with the sum of the K per-position draft losses and the rank-head cross-entropy $\mathcal{L}^{\text{rank}}$, both masked by $m_{t,k}$,

$$\mathcal{L} \;=\; \sum_{k=1}^{K} \mathcal{L}_k^{\text{draft}} \;+\; \mathcal{L}^{\text{rank}}. \tag{7}$$

3.4 Cost-aware serving-time adaptation

The training procedure above yields a fixed drafter, but accepted length degrades when the serving prompt distribution shifts. Refreshing the drafter at serving time can restore the lost accepted length, but each backward roughly costs as much as one target forward, so an indiscriminate schedule negates the throughput it tries to protect. We therefore answer two questions per query: whether to update, and which parameters to update. The verifier's output provides a free signal for the first, and the drafter's modular architecture provides the action structure for the second.

Verifier-derived update signal. The drafter's distribution and the target's chosen token are both available at every rejected position, so reading them requires no extra work. For each rejected position k on a verified path, let $r_{t,k} \in [0, 1]$ be the drafter's probability of the target's chosen token. We aggregate these into a query-level signal

$$s \;=\; \sum_{k \,\in\, \text{rejected}} (1 - r_{t,k}), \tag{8}$$

which is large when the drafter is far from the target's choices at multiple rejected positions and small when the two are nearly aligned.

Action set and per-query selection. A bandit selects per query among three actions, each addressing a different drafter-error mode: skip when the drafter is already well-calibrated, head-only when sound internal representations are mis-mapped by the lm_head and rank head, and full update when the decoder trunk itself is mismatched. Let s_{trig} denote the value of s at the query that triggered an update, and let v_{head} and v_{full} each be an exponentially-weighted moving average (EWMA) of the realized reward $\Delta_{\text{tp}}^{\text{observed}}/s_{\text{trig}}$. Because throughput is measured over the interval that follows an update, v_{action} already nets out the update's own cost. At query time we predict the net throughput gain of each non-skip action as

$$\widehat{\Delta}_{\text{tp}}(\text{action}) \;=\; s \cdot v_{\text{action}}. \tag{9}$$

We pick the action with the largest predicted gain, and we skip if both predictions are non-positive. The measurement interval spans the N queries following an update, and at its close the EWMA is revised as

$$v_{\text{action}} \;\leftarrow\; (1-\alpha)\, v_{\text{action}} \;+\; \alpha\, \frac{\Delta_{\text{tp}}^{\text{observed}}}{s_{\text{trig}}}, \tag{10}$$

with $\alpha = 0.10$. The revision is skipped when $s_{\text{trig}} < s_{\min}$, since dividing by a small s_{trig} amplifies noise in the throughput estimate.

Asynchronous updates and drift control. The drafter is held in two copies at serving time, θ_{inf} and θ_{train}. The bandit's chosen update is applied to θ_{train} on a separate stream, so drafting on θ_{inf} is not blocked. The trained copy is periodically copied back into θ_{inf}. To prevent the drafter from drifting away from the pre-deployment distribution under repeated updates, we add a KL penalty $\lambda\,\mathrm{KL}(p_\theta \,\|\, p_{\theta_0})$ against the pre-deployment drafter θ_0 to the per-update objective, with $\lambda = 0.01$. As a fail-safe we additionally monitor accepted length over a sliding window of three consecutive serving intervals, and if it decreases monotonically across all three we revert θ_{inf} to the most recent good checkpoint and reset the bandit's value estimates.

These four mechanisms together target the throughput formula $\Phi = \tau/(T_{\mathcal{M}} + T_{\mathcal{D}})$ from complementary angles. The block forward and the layer-wise shift cap $T_{\mathcal{D}}$ while preserving τ within a block. Rank-guided branching and valid-prefix training jointly lift τ at inference. Serving-time adaptation guards τ against deployment shift.

4 Experiments

4.1 Setup

We evaluate SpecBlock on three target models, Llama-3.1-8B-Instruct [38], Qwen3-8B, and Qwen3-32B [39]. The drafter for each target is a stack of $L{=}2$ Transformer decoder layers matching the target's per-layer architecture, with per-block depth $K{=}4$ and $M{=}2$ blocks per inference iteration. We measure on a single NVIDIA A100-80GB GPU at batch size 1 with temperature 0 and 1.0, with cost-aware adaptation in its single-GPU variant unless noted. We report speedup (Spd) over vanilla autoregressive decoding, throughput Φ in tokens per second, accepted length τ averaged over verifier calls, and drafting cost $T_{\mathcal{D}}\%$ as the share of per-iteration latency spent on drafter forwards.

Baselines. We compare against representative drafters of several kinds, all reproduced under the same target model, attention backend, and tree-size budget. Vanilla decoding without speculation anchors the speedup. Standard speculative sampling (SpS) [1, 2] uses a pretrained smaller model from the same family as the drafter, with no further training. We pair Llama-3.1-8B with Llama-3.2-1B, and the Qwen3 targets with Qwen3-0.6B. Among autoregressive drafters we use EAGLE-3 [9]. Among parallel drafters we use Medusa [10] and ParallelSpec [18]. Falcon [23] is the closest blockwise prior work. For online verifier-feedback adaptation, Online Speculative Decoding (OSD) [34] is instantiated on the SpecBlock drafter as SpecBlock+OSD.

Training. All trainable drafters are trained on prompts from UltraChat-200K[3] [40] and ShareGPT,[4] with answers regenerated by the target model so that the training distribution matches what the target actually emits at inference. Training runs for 20 epochs with AdamW (learning rate $5{\times}10^{-5}$,

[3] https://huggingface.co/datasets/HuggingFaceH4/ultrachat_200k
[4] https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered

Table 1: Speedup (Spd) over vanilla decoding and average accepted length τ per benchmark at A100-80GB, batch size 1, under HuggingFace Transformers. "$T_{\mathcal{D}}\%$" is the per-method drafting-cost share. Bold indicates the best speedup within each model group. SpecBlock+adapt is SpecBlock with cost-aware serving-time adaptation. Dashes indicate query streams too short to offset the cost of online updates.

Model	Method	HumanEval		MATH-500		Alpaca		NQ		MT-Bench		WMT-23		Mean		
		Spd	τ	Spd	τ	Spd	τ	Spd	τ	Spd	τ	Spd	τ	Spd	τ	$T_{\mathcal{D}}\%$
Temperature = 0																
Llama-3.1-8B	SpS	1.55×	2.91	1.17×	2.29	1.48×	2.70	1.06×	2.09	1.38×	2.45	0.99×	1.92	1.27×	2.39	38
	Medusa	2.16×	2.70	1.53×	2.30	2.01×	2.44	1.44×	2.00	1.88×	2.59	1.34×	2.05	1.73×	2.35	6
	ParallelSpec	2.50×	3.16	1.89×	2.64	2.39×	3.28	1.78×	2.63	2.36×	3.21	1.66×	2.30	2.10×	2.87	8
	Falcon	3.06×	4.69	2.30×	3.74	2.86×	4.23	2.12×	3.38	2.66×	4.31	1.90×	3.33	2.48×	3.95	26
	EAGLE-3	3.59×	6.98	2.66×	5.93	3.35×	6.27	2.42×	5.39	**3.22**×	6.16	2.28×	4.60	2.92×	5.89	31
	SpecBlock	**3.92**×	5.16	**3.07**×	4.03	**3.40**×	4.66	**3.00**×	4.40	3.10×	4.46	**2.79**×	3.75	**3.21**×	4.41	16
	SpecBlock+OSD	—	—	3.10×	4.24	3.43×	4.74	3.25×	5.41	—	—	2.80×	3.85	3.14×	4.56	18
	SpecBlock+adapt	—	—	**3.14**×	4.23	**3.47**×	4.72	**3.51**×	5.41	—	—	**2.81**×	3.81	**3.24**×	4.54	15
Qwen3-8B	EAGLE-3	2.45×	4.53	1.94×	4.46	2.35×	4.29	**2.50**×	4.20	**2.40**×	4.23	1.81×	3.12	2.24×	4.14	29
	SpecBlock	**2.50**×	3.59	**2.53**×	3.71	**2.58**×	3.73	2.26×	3.21	2.33×	3.32	**2.30**×	3.28	**2.42**×	3.47	14
	SpecBlock+OSD	—	—	2.57×	3.92	2.62×	3.80	2.49×	3.75	—	—	2.32×	3.37	2.50×	3.71	19
	SpecBlock+adapt	—	—	**2.61**×	3.90	**2.64**×	3.78	**2.69**×	3.75	—	—	**2.34**×	3.36	**2.56**×	3.70	17
Qwen3-32B	EAGLE-3	2.54×	4.36	1.82×	4.27	2.28×	4.09	**2.38**×	3.96	2.18×	4.09	1.71×	2.96	2.15×	3.96	24
	SpecBlock	**2.61**×	3.47	**2.48**×	3.53	**2.48**×	3.54	2.17×	3.07	**2.21**×	3.21	**2.20**×	3.15	**2.37**×	3.33	11
	SpecBlock+OSD	—	—	2.51×	3.72	2.50×	3.62	2.36×	3.61	—	—	2.24×	3.24	2.40×	3.55	15
	SpecBlock+adapt	—	—	**2.55**×	3.73	**2.51**×	3.61	**2.57**×	3.61	—	—	**2.24**×	3.22	**2.47**×	3.54	13
Temperature = 1.0																
Llama-3.1-8B	SpS	1.33×	2.32	0.58×	1.87	1.06×	2.33	0.70×	1.76	0.63×	2.01	0.82×	1.56	0.85×	1.97	40
	Medusa	1.85×	2.20	0.83×	1.90	1.50×	2.07	1.01×	1.67	0.90×	2.13	1.10×	1.63	1.20×	1.93	7
	ParallelSpec	2.27×	2.65	0.97×	2.18	1.78×	2.85	1.20×	2.20	1.06×	2.67	1.33×	1.84	1.43×	2.40	9
	Falcon	2.64×	3.86	1.15×	3.01	2.11×	3.64	1.44×	2.77	1.27×	3.55	1.51×	2.83	1.69×	3.28	27
	EAGLE-3	2.98×	6.09	1.32×	4.31	2.36×	5.17	1.66×	4.27	1.48×	4.25	1.81×	3.94	1.94×	4.67	30
	SpecBlock	**3.21**×	4.62	**1.74**×	3.24	**2.57**×	4.02	**1.92**×	3.60	**1.50**×	3.46	**2.31**×	3.41	**2.20**×	3.73	14
	SpecBlock+OSD	—	—	1.77×	3.38	2.66×	4.25	2.17×	4.59	—	—	2.36×	3.59	2.24×	3.95	17
	SpecBlock+adapt	—	—	**1.78**×	3.31	**2.72**×	4.22	**2.39**×	4.57	—	—	**2.36**×	3.57	**2.31**×	3.92	15
Qwen3-8B	EAGLE-3	2.32×	4.34	2.18×	4.28	1.94×	4.14	**1.94**×	4.09	1.93×	3.93	1.65×	3.10	1.99×	3.98	29
	SpecBlock	**2.44**×	3.53	**2.60**×	3.63	**2.06**×	3.59	1.81×	3.06	**2.06**×	3.19	**2.11**×	3.19	**2.18**×	3.37	14
	SpecBlock+OSD	—	—	2.64×	3.83	2.14×	3.72	2.06×	3.55	—	—	2.16×	3.34	2.25×	3.61	19
	SpecBlock+adapt	—	—	**2.67**×	3.79	**2.19**×	3.69	**2.27**×	3.55	—	—	**2.19**×	3.31	**2.33**×	3.59	18
Qwen3-32B	EAGLE-3	2.17×	4.21	2.02×	4.06	1.84×	3.94	**1.87**×	3.95	1.85×	3.82	1.55×	2.95	1.88×	3.82	25
	SpecBlock	**2.26**×	3.43	**2.48**×	3.48	**1.94**×	3.42	1.75×	2.92	**1.99**×	3.09	**2.07**×	3.10	**2.07**×	3.24	11
	SpecBlock+OSD	—	—	2.51×	3.67	2.00×	3.55	1.99×	3.41	—	—	2.10×	3.23	2.15×	3.47	14
	SpecBlock+adapt	—	—	**2.55**×	3.64	**2.06**×	3.53	**2.19**×	3.40	—	—	**2.14**×	3.22	**2.24**×	3.45	13

cosine schedule, gradient clip 0.5). For SpecBlock, the valid-prefix mask and the cross-block training procedure of §3.3 are used throughout, and the rank head is enabled after the first 2,000 update steps to let the drafter trunk reach a stable distribution before bucket supervision.

Evaluation tasks. We evaluate on six benchmarks spanning conversation, code, competition math, instruction following, question answering, and translation: MT-Bench [41], HumanEval [42], MATH-500 [43], Alpaca [44], Natural Questions (NQ) [45], and WMT-23 [46].

4.2 Main results

Table 1 reports per-benchmark speedup over vanilla decoding and accepted length τ on three target models at A100-80GB, batch size 1. Among static drafters, SpecBlock improves mean speedup over EAGLE-3 by 8–13% across all six configurations. Cost-aware adapt further lifts speedup over the always-update SpecBlock+OSD by 2–4% on the four benchmarks where the bandit engages. HumanEval at 164 prompts and MT-Bench at 80 prompts are too short for the acceptance gain to amortize the backward cost of adaptation, so we do not evaluate adapt on them. Target-verifier time is essentially constant across these methods at about 37 ms per iteration on Llama-3.1-8B, so throughput reduces to how each drafter trades τ against drafter cost $T_{\mathcal{D}}$.

Among static drafters, EAGLE-3 reaches the highest τ=5.89 through seven sequential drafter calls but pays a 31% drafter share, while parallel drafters such as Medusa and ParallelSpec cut this to 6–9% at the cost of capping τ at around 2 to 3. SpecBlock condenses drafting into two block

Table 2: Ablations on Llama-3.1-8B at $T{=}0$, removing one component of the base architecture (top) or of cost-aware serving-time adaptation (bottom) per row.

	HumanEval		MATH-500		Alpaca		NQ		MT-Bench		WMT-23		Mean	
Variant	Spd	τ	Spd	τ	Spd	τ	Spd	τ	Spd	τ	Spd	τ	Spd	τ
Base architecture														
SpecBlock	3.92×	5.16	3.07×	4.03	3.40×	4.66	3.00×	4.40	3.10×	4.46	2.79×	3.75	3.21×	4.41
− prefix broadcast	3.69×	4.96	2.95×	3.73	3.08×	4.32	2.84×	4.22	2.83×	4.08	2.57×	3.49	2.99×	4.13
− layer-wise shift	3.65×	4.85	2.79×	3.55	3.01×	4.13	2.87×	4.13	2.78×	3.97	2.62×	3.49	2.95×	4.02
− valid-prefix curriculum	3.76×	5.06	2.94×	3.94	3.27×	4.53	2.86×	4.15	2.87×	4.15	2.69×	3.56	3.06×	4.23
− rank-guided branching	3.80×	4.84	2.96×	3.87	3.40×	4.68	2.92×	4.32	3.00×	4.26	2.69×	3.54	3.13×	4.25
Cost-aware serving-time adaptation														
SpecBlock+adapt	—	—	3.14×	4.23	3.47×	4.72	3.51×	5.41	—	—	2.81×	3.81	3.24×	4.54
− cost-aware bandit	—	—	2.96×	4.35	3.41×	4.81	3.48×	5.26	—	—	2.74×	3.75	3.14×	4.54
− head-only action	—	—	2.91×	4.25	3.20×	4.97	3.31×	5.46	—	—	2.60×	3.65	3.00×	4.58

forwards, dropping drafter time from 17 ms to 7 ms while the layer-wise shift retains $\tau{=}4.41$ on Llama-3.1-8B, only 1.48 tokens below EAGLE-3 despite the much shorter chain. Cost-aware adapt and SpecBlock+OSD reach essentially the same τ, within 0.02 tokens, but SpecBlock+adapt holds $T_{\mathcal{D}}\%$ at 13–18 versus OSD's 14–19: the bandit skips weak signals and routes most updates to the head-only action, whose backward over the lm_head and rank head is more than an order of magnitude cheaper than a full-drafter backward. At $T{=}0$, single-domain benchmarks such as NQ and MATH-500 lift τ by 0.2 to 1.0 as streaming queries share a stable target, while mixed-instruction Alpaca lifts only 0.05 to 0.07 because gradients across instruction types partially cancel.

The trade-off strengthens on larger targets: drafter share falls from 31% to 24% for EAGLE-3 between Llama-3.1-8B and Qwen3-32B, and from 16% to 11% for SpecBlock, narrowing SpecBlock's relative drafting cost from 52% to 46% of EAGLE-3's. Sampling at $T{=}1.0$ reduces τ across all methods because rejection sampling against a high-entropy target accepts fewer draft tokens, but the relative ordering is unchanged.

4.3 Ablations

To evaluate each design component, Table 2 removes one component per row on Llama-3.1-8B at $T{=}0$, with the base group on the six benchmarks and the adaptation group on the four where the bandit engages. Base rows ablate the prefix broadcast, the layer-wise shift, the valid-prefix curriculum, and the rank-guided branching, with the last replaced by a uniform fixed-k tree at the same node budget. Adaptation rows ablate the cost-aware bandit, leaving an always-update policy, and the head-only action, forcing every triggered update to a full-drafter backward.

Layer-wise shift contributes the largest single-component gain in mean speedup: removing it drops Spd from $3.21\times$ to $2.95\times$ and τ from 4.41 to 4.02, since cross-position causal attention alone fails to retain within-block dependence at deeper positions. The drop is heaviest on long-form benchmarks where deeper-position acceptance determines chain length, with Alpaca losing $0.39\times$ and MT-Bench $0.32\times$, while shorter-answer NQ loses only $0.13\times$. Removing the prefix broadcast costs $0.22\times$ as the position queries alone cannot anchor the K positions to the verified prefix. The valid-prefix curriculum and rank-guided branching add $0.15\times$ and $0.08\times$ respectively by preventing wrong-prefix supervision and reallocating verifier budget to uncertain positions.

On the adaptation side, removing the cost-aware bandit drops mean Spd by $0.10\times$ while leaving τ unchanged at 4.54, since always-update reaches the same accepted length but pays a backward every eight queries. Removing the head-only action drops mean Spd by $0.24\times$ with τ slightly higher at 4.58, as forcing every triggered update to a full backward gains only 0.04 in τ at the cost of an order-of-magnitude longer backward. The cost-aware bandit and head-only action therefore keep adaptation cheap by skipping weak signals and routing most updates to a backward over only the lm_head and rank head.

5 Conclusion

In this paper, we introduce SpecBlock, a block-iterative drafter that produces K dependent positions per forward and extends the path through repeated block expansions starting from earlier positions' hidden states. A layer-wise shift carries each previous position's hidden state into every decoder layer to preserve within-block dependence, a co-trained rank head sets per-position branching to allocate verifier budget where the drafter is uncertain, and a valid-prefix curriculum masks the loss after the first wrong prediction in the block to prevent wrong prefixes from interfering with training. Beyond static drafting, a cost-aware bandit at deployment uses free verifier feedback to update the drafter selectively, only when the expected throughput gain exceeds the update cost. SpecBlock improves mean speedup over EAGLE-3 by 8–13% across three target models, with cost-aware adaptation extending the gain to 11–19%.

Acknowledgments and Disclosure of Funding

The research work described in this paper was supported by Guangzhou HKUST Fok Ying Tung Research Institute (grant #2023ZD007), HKUST-MetaX Joint Laboratory for Advanced AI Computing (grant#METAX24EG01-G). It was partially conducted in JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust. This work is also supported by the National Natural Science Foundation of China (Grant No. 62102277) and the National Key R&D Program of China (Grant No. 2022YFC3303600).

References

[1] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[2] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[3] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, page 932–949. ACM, April 2024.

[4] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. volume 36, pages 30222–30242, 2023.

[5] Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2023.

[6] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. volume 37, pages 129531–129563, 2024.

[7] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. 2024.

[8] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. pages 7421–7432, 2024.

[9] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

[10] 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.

[11] Lefan Zhang, Xiaodan Wang, Yanhua Huang, and Ruiwen Xu. Learning harmonized representations for speculative sampling. *arXiv preprint arXiv:2408.15766*, 2024.

[12] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. pages 11263–11282, 2024.

[13] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*, 2024.

[14] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. Rest: Retrieval-based speculative decoding. pages 1582–1595, 2024.

[15] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024.

[16] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. volume 31, 2018.

[17] Feng Lin, Hanling Yi, Yifan Yang, Hongbin Li, Xiaotian Yu, Guangming Lu, and Rong Xiao. Bita: Bi-directional tuning for lossless acceleration in large language models. *Expert Systems with Applications*, 279:127305, 2025.

[18] Zilin Xiao, Hongming Zhang, Tao Ge, Siru Ouyang, Vicente Ordonez, and Dong Yu. Parallelspec: Parallel drafter for efficient speculative decoding. *arXiv preprint arXiv:2410.05589*, 2024.

[19] Zihao An, Huajun Bai, Ziqiong Liu, Dong Li, and Emad Barsoum. Pard: Accelerating llm inference with low-cost parallel draft model adaptation. *arXiv preprint arXiv:2504.18583*, 2025.

[20] Fuliang Liu, Xue Li, Ketai Zhao, Yinxi Gao, Ziyan Zhou, Zhonghui Zhang, Zhibin Wang, Wanchun Dou, Sheng Zhong, and Chen Tian. Dart: Diffusion-inspired speculative decoding for fast llm inference. *arXiv preprint arXiv:2601.19278*, 2026.

[21] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. *arXiv preprint arXiv:2402.05109*, 2024.

[22] Haiduo Huang, Jiangcheng Song, Wenzhe Zhao, and Pengju Ren. Fasteagle: Cascaded drafting for accelerating speculative decoding. pages 4111–4115, 2026.

[23] Xiangxiang Gao, Weisheng Xie, Yiwei Xiang, and Feng Ji. Falcon: Faster and parallel inference of large language models through enhanced semi-autoregressive drafting and custom-designed decoding tree. 39(22):23933–23941, 2025.

[24] Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. Pearl: Parallel speculative decoding with adaptive draft length. *arXiv preprint arXiv:2408.11850*, 2024.

[25] Taehyeon Kim, Ananda Theertha Suresh, Kishore Papineni, Adrian Benton, and Michael Riley. Exploring and improving drafts in blockwise parallel decoding. *arXiv preprint arXiv:2404.09221*, 2024.

[26] Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set block decoding is a language model inference accelerator. *arXiv preprint arXiv:2509.04185*, 2025.

[27] Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent drafter for fast speculative decoding in large language models. *arXiv preprint arXiv:2403.09919*, 2024.

[28] Feiye Huo, Jianchao Tan, Kefeng Zhang, Xunliang Cai, and Shengli Sun. C2t: A classifier-based tree construction method in speculative decoding. *arXiv preprint arXiv:2502.13652*, 2025.

[29] Jikai Wang, Yi Su, Juntao Li, Qingrong Xia, Zi Ye, Xinyu Duan, Zhefeng Wang, and Min Zhang. Opt-tree: Speculative decoding with adaptive draft tree structure. *Transactions of the Association for Computational Linguistics*, 13:188–199, 2025.

[30] Yunfan Xiong, Ruoyu Zhang, Yanzeng Li, and Lei Zou. Dyspec: Faster speculative decoding with dynamic token tree structure. *World Wide Web*, 28(3):36, 2025.

[31] Tianyu Liu, Qitan Lv, Yuhao Shen, Xiao Sun, and Xiaoyan Sun. Talon: Confidence-aware speculative decoding with adaptive token trees. *arXiv preprint arXiv:2601.07353*, 2026.

[32] Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent YF Tan, and Zhuoran Yang. Banditspec: Adaptive speculative decoding via bandit algorithms. *arXiv preprint arXiv:2505.15141*, 2025.

[33] Kaixuan Huang, Xudong Guo, and Mengdi Wang. SpecDec++: Boosting speculative decoding via adaptive candidate lengths. In *Conference on Language Modeling*, 2024.

[34] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Alvin Cheung, Zhijie Deng, Ion Stoica, and Hao Zhang. Online speculative decoding. 2023.

[35] Shrenik Bhansali and Larry Heck. Draft, verify, and improve: Toward training-aware speculative decoding. *arXiv preprint arXiv:2510.05421*, 2025.

[36] Jiyoung Park, Hankyu Jang, Changseok Song, and Wookeun Jung. Tide: Temporal incremental draft engine for self-improving llm inference. *arXiv preprint arXiv:2602.05145*, 2026.

[37] Junxiong Wang, Fengxiang Bie, Jisen Li, Zhongzhu Zhou, Zelei Shao, Yubo Wang, Yinghui Liu, Qingyang Wu, Avner May, Sri Yanamandra, et al. When rl meets adaptive speculative training: A unified training-serving system. *arXiv preprint arXiv:2602.06932*, 2026.

[38] 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.

[39] 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.

[40] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3029–3051, 2023.

[41] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 2023.

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

[43] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[44] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

[45] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[46] Tom Kocmi, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Thamme Gowda, Yvette Graham, Roman Grundkiewicz, Barry Haddow, et al. Findings of the 2022 conference on machine translation (wmt22). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 1–45, 2022.

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

[48] Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. *arXiv preprint arXiv:2607.05147*, 2026.

A Implementation Details

A.1 Drafter setup and training configuration

The drafter follows the target's decoder-layer design and operates at the same hidden dimension. The decoder and LM head are trained from scratch, while the token embedding is initialised from the target and frozen during training. Following EAGLE-3, the drafter operates over a reduced vocabulary of the 32,000 most frequent tokens, accounting for 98.7% of training tokens.

We use AdamW with PyTorch defaults $(\beta_1, \beta_2)=(0.9, 0.999)$ and a global batch size of 96. The cosine schedule warms up linearly over the first 1.5% of total update steps. Training takes approximately 3,000 A100-80GB GPU-hours per drafter. Inference runs on a single A100-80GB GPU by default, and cost-aware adaptation additionally has a dual-GPU variant.

A.2 Attention Mask for SpecBlock

Figure 3 illustrates the attention pattern across one cross-block iteration, with $B_{i,k}$ denoting the k-th draft position of block i. Each drafter forward attends to three sources of context. The verified prefix contributes the K keys and values that every previously committed position produced in its own draft forward, all fully visible to the current forward. Preceding blocks within the current verifier iteration are visible only along the branch path. When a new block starts from position j of a preceding block, positions $0, \ldots, j$ are on-path and visible while later positions are off-path and masked. Within the current block, attention is causal so position k attends only to positions $\leq k$.

A.3 Cost-aware adaptation

Bandit lifecycle. A short warmup of 10 queries collects the baseline throughput before any training event. The bandit then enters a cold-start phase of 8 events with ε-greedy exploration, with ε decaying linearly from 0.30 to 0.10, so the value estimates v_{head} and v_{full} are seeded from each action. After each train event, the next 2 queries are blocked from triggering further updates, and the EWMA update on the previous mode's reward is skipped when its trigger signal $s_{\text{trig}} < 5$ to avoid amplifying noise from low-signal updates.

Scheduling and synchronisation. Figure 4 illustrates the per-query schedule. Training fires on the train stream only when its buffer saturates, and is timed so the backward does not interfere with the verifier. Weight synchronisation between θ_{train} and θ_{inf} is timed during target-verify windows when the drafter is otherwise idle. In single-GPU deployment the train stream runs on the same device as inference but uses a separate CUDA stream, so its kernels share the GPU's streaming multiprocessors

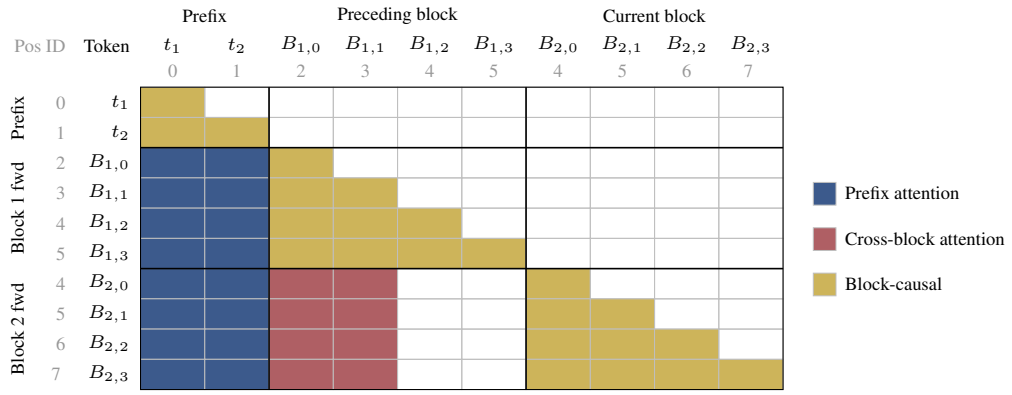


Figure 3: Attention pattern across one cross-block iteration with $K=4$. Block 2 branches from $B_{1,1}$, so the path leading to block 2 consists of $B_{1,0}$ and $B_{1,1}$. The prefix's tokens attend to themselves causally. Each block forward sees the verified prefix as Prefix attention plus itself as Block-causal. Block 2's queries additionally attend to the on-path positions $B_{1,0}, B_{1,1}$ as Cross-block attention, while the off-path positions $B_{1,2}, B_{1,3}$ are masked.

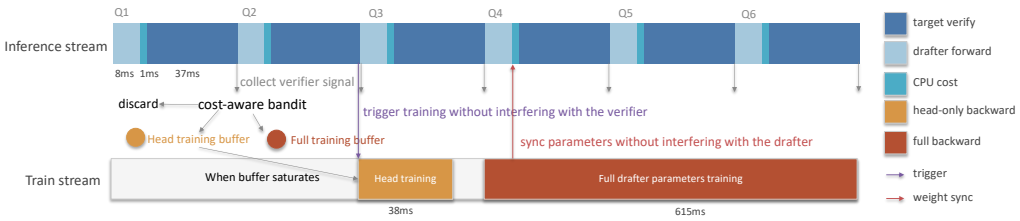


Figure 4: Cost-aware adaptation scheduling. The cost-aware bandit ingests the verifier signal at every query end and routes it into the head or full training buffer. When a buffer saturates, the corresponding backward fires on the train stream and weights sync back, both timed to avoid interfering with the verifier and the drafter. The schedule applies to single-GPU and dual-GPU deployments.

Algorithm 1 Cost-aware adaptation, per query.

Require: pre-deployment drafter θ_0, two-copy state $(\theta_{\text{inf}}, \theta_{\text{train}})$, value estimates $(v_{\text{head}}, v_{\text{full}})$, head and full buffers $(\mathcal{B}_{\text{head}}, \mathcal{B}_{\text{full}})$, EWMA decay α, KL weight λ, signal threshold $s_{\min}$, measurement interval length N

complete drafting and verification on θ_{inf}
$s \leftarrow \sum_{k \in \text{rejected}} (1 - r_{t,k})$ ▷ verifier-derived signal
$\widehat{\Delta}_{\text{tp}}(\text{head}) \leftarrow s \cdot v_{\text{head}}$; $\widehat{\Delta}_{\text{tp}}(\text{full}) \leftarrow s \cdot v_{\text{full}}$
if $\widehat{\Delta}_{\text{tp}}(\text{head}) \le 0$ **and** $\widehat{\Delta}_{\text{tp}}(\text{full}) \le 0$ **then**
  $a \leftarrow$ skip
else
  $a \leftarrow \arg\max_{\text{action} \in \{\text{head,full}\}} \widehat{\Delta}_{\text{tp}}(\text{action})$
end if
if $a \neq$ skip **then**
  push (rejected positions, target distributions, s) into $\mathcal{B}_a$
end if
if $\mathcal{B}_a$ saturates **then**
  $s_{\text{trig}} \leftarrow s$; $a_{\text{trig}} \leftarrow a$ ▷ snapshot trigger signal and action
  on train stream: step θ_{train} on $\mathcal{L}_a^{\text{adapt}} + \lambda\, \text{KL}(p_{\theta_{\text{train}}} \,\|\, p_{\theta_0})$
  clear $\mathcal{B}_a$
  during the next target-verify window: $\theta_{\text{inf}} \leftarrow \theta_{\text{train}}$
  open measurement interval of N queries to record $\Delta_{\text{tp}}^{\text{observed}}$
end if
if measurement interval closes **and** $s_{\text{trig}} \ge s_{\min}$ **then**
  $v_{a_{\text{trig}}} \leftarrow (1 - \alpha)\, v_{a_{\text{trig}}} + \alpha\, \Delta_{\text{tp}}^{\text{observed}} / s_{\text{trig}}$ ▷ EWMA update
end if
if accepted length decreases monotonically over the last 3 windows **then**
  revert θ_{inf} to the last good checkpoint, reset $v_{\text{head}}, v_{\text{full}}$
end if

with inference rather than preempting it. In dual-GPU deployment the train stream lives on a second device and the two drafter copies sync via periodic in-place weight copy.

Per-query control flow. Algorithm 1 ties together the bandit, the train stream, and the drift control. The verifier-derived signal predicts the net gain of each non-skip action at query time, and its value s_{trig} at the query that fires an update later gates the EWMA update of that update's reward. When a buffer saturates, the corresponding backward fires on the train stream, and the trained weights are synced into θ_{inf} during the next target-verify window. A monotonic drop in accepted length over three windows triggers a rollback to the last good checkpoint.

A.4 Baselines and benchmarks

Baselines. We compare against six drafting baselines.

- **Standard speculative sampling (SpS)** [1, 2] samples a chain of future tokens autoregressively from a smaller off-the-shelf drafter and lets the target verify the chain in one parallel forward. We pair Llama-3.1-8B with Llama-3.2-1B, and the Qwen3 targets with Qwen3-0.6B.
- **Medusa** [10] attaches K independent decoding heads at fixed offsets to the target's last hidden state, with no cross-position attention or layer-wise dependence.
- **ParallelSpec** [18] predicts K future tokens in one drafter forward by appending K learnable `[MASK]` tokens after the prefix and reading their last-layer hidden states, under a group-wise causal mask that blocks attention from `[MASK]`s in earlier parallel groups.
- **Falcon** [23] is the closest blockwise prior work. It drafts semi-autoregressive blocks of k tokens with a hybrid drafter combining LSTM layers with relaxed-causal-mask self-attention, letting positions inside the same $k \times k$ block attend to one another, and verifies through a custom-designed static decoding tree.
- **EAGLE-3** [9] is an autoregressive token-level drafter that fuses the target's low-, mid-, and high-level hidden states as input, replacing the top-layer-only reuse of EAGLE-1/2 [7, 8], and grows a dynamic draft tree depth by depth by calling the drafter once per added depth.
- **SpecBlock+OSD** instantiates Online Speculative Decoding [34] on the SpecBlock drafter as the always-update baseline. Rejection-position pairs of the draft and target distributions are logged to a replay buffer, and the drafter is updated every eight queries via forward-KL distillation, with no bandit gating over which signals to apply.

Benchmarks. The six benchmarks vary substantially in prompt count, from 80 multi-turn dialogues in MT-Bench and 164 code prompts in HumanEval, through 500 competition problems in MATH-500 and 549 translation pairs in WMT-23, up to 4,000 instruction prompts in Alpaca and 3,610 open-domain questions in Natural Questions. Each prompt is rendered through the target's official chat template, with thinking mode disabled for Qwen3 so the verifier output is comparable to Llama, and passed once through the speculative decoding loop under greedy decoding ($T{=}0$) or stochastic decoding ($T{=}1.0$). Generation is capped at 1,024 newly committed tokens. We report speedup as the wall-clock ratio over vanilla autoregressive decoding on the same prompt set and accepted length τ as the average length committed per verifier call. We compute $T_{\mathcal{D}}\% = 100\, t_{\text{draft}}/(t_{\text{draft}} + t_{\text{target}} + t_{\text{verify}})$. The three terms measure candidate construction, target-model forward passes, and verification overhead, respectively. Cost-aware adaptation is reported only on MATH-500, Alpaca, NQ, and WMT-23, whose streaming-prompt counts are large enough for the adaptation backward to amortize its cost; HumanEval and MT-Bench leave too little traffic for the bandit's value estimates to converge.

A.5 Inference procedure

Algorithm 2 traces one verifier iteration of SpecBlock. The drafter is invoked at most M times to grow a tree of up to $M{\cdot}K$ depth, the verifier scores all candidates in a single parallel forward, and the longest accepted prefix is committed before the next iteration starts. The first block conditions on the target's multi-layer features at the prefix-end position, while later blocks condition on the drafter's own cached last-layer state at the starting position, bypassing the projection W_{cond}. Per-position branching width follows the rank head's bucket through the map $b(\cdot)$. We recommend two configurations of $b(\cdot)$ over the four buckets defined in §3.2: $[2, 4, 10, 0]$, which concentrates tree budget on the less confident buckets where the target token sits deeper and gives up on the rank$>$10 bucket, and $[2, 4, 6, 4]$, which trims the wide rank-5–10 bucket and keeps a 4-candidate fallback at give-up positions for out-of-distribution traffic.

B Rank head

Distribution summary features. Section 3.2 states that the rank head reads a 15-dimensional summary $\psi(p_{t,k})$ of the draft distribution alongside the hidden state. We detail those 15 dimensions here. The summary captures how peaky or flat the distribution is through three kinds of signal. The bulk of ψ records the log-probability profile of the top-10 tokens, which describes how mass spreads across the most likely candidates and contributes ten dimensions. Three further dimensions capture the logit gaps between the top token and its rank-2, rank-3, and rank-5 competitors, telling the rank

Algorithm 2 SpecBlock inference, one verifier iteration.

Require: target $\mathcal{M}$, drafter $\mathcal{D}_\theta$, last verified token x_t and multi-layer features $(h_t^{\text{low}}, h_t^{\text{mid}}, h_t^{\text{top}})$ from $\mathcal{M}$'s most recent forward, block depth K, block budget M, bucket-to-branching map $b(\cdot)$
Ensure: tokens committed in this iteration

$c_t \leftarrow W_{\text{cond}}\,[h_t^{\text{low}}, h_t^{\text{mid}}, h_t^{\text{top}}]$
$\text{starts} \leftarrow \{(c_t,\, \text{embed}(x_t))\}$ ▷ first-block starting point uses target features
$\text{tree} \leftarrow$ empty draft tree rooted at x_t
for $m = 1$ **to** M **do**
  $\{p_{i,k},\, h_{i,k}^{(L)},\, \text{bkt}_{i,k}\}_{i,k=1}^{K} \leftarrow \mathcal{D}_\theta(\text{starts})$ ▷ batched forward, K positions per starting point
  for each starting point i and each position k **do**
    attach the top-$b(\text{bkt}_{i,k})$ tokens of $p_{i,k}$ to tree as candidates at position (i, k)
  end for
  $\text{starts} \leftarrow$ next-block starting points selected by the rank head, each carrying its cached $h_{i,k}^{(L)}$ and sampled token
end for
$(\text{prefix},\, \text{bonus}) \leftarrow \mathcal{M}.\text{verify}(\text{tree})$ ▷ single parallel target forward
return prefix $\cup$ bonus

Table 3: Rank head classification quality on $\sim$ 72,000 held-out positions of the SpecBlock drafter for Llama-3.1-8B.

Bucket	Frequency (%)	Precision	Recall	F1
b_0 (rank $=1$)	75.2	0.978	0.687	0.807
b_1 (rank $\in [2, 4]$)	15.0	0.528	0.476	0.501
b_2 (rank $\in [5, 10]$)	4.1	0.468	0.448	0.458
b_3 (rank > 10)	5.7	0.854	0.792	0.822

head how distinguishable the leader is from close runners-up. The remaining two dimensions are scalar summaries: the probability of the top token, and the entropy of the distribution.

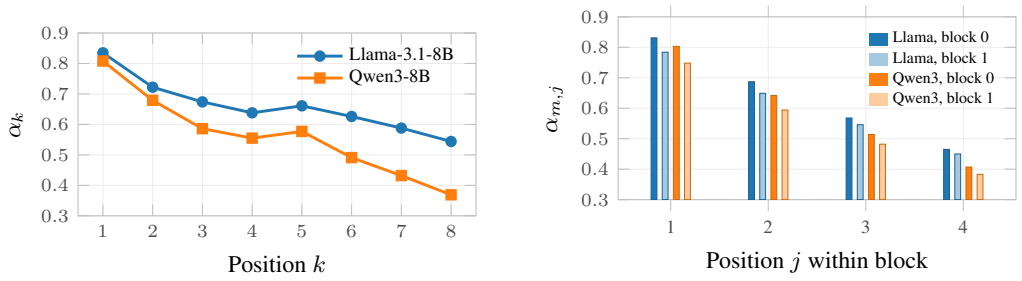
Classification quality. The rank head is supervised as a four-way classifier over the bucket labels defined in §3.2, and its classification quality directly affects how the verifier budget is spent. We evaluate it as a standalone classifier on $\sim$ 72,000 validation positions of the SpecBlock drafter for Llama-3.1-8B, restricted to positions whose valid-prefix mask is one. At each such position we record the predicted bucket and the ground-truth bucket derived from the target token's rank within $p_{t,k}$.

Table 3 reports per-bucket precision, recall, and F1 together with class frequency. The class frequencies are sharply imbalanced, with b_0 accounting for 75.2% of positions while b_2 and b_3 each fall under 6%. The two extreme buckets are the easiest to classify. The confident bucket b_0 reaches precision 0.978 since the drafter is well-calibrated when the target sits at rank 1, and the give-up bucket b_3 reaches F1 0.822 since high-rank positions carry clear distribution-shape signals such as flat or multi-modal $p_{t,k}$. The two middle buckets b_1 and b_2 are the hardest, with F1 around 0.46–0.50, and most errors fall between the two adjacent buckets.

C Per-position acceptance rate

We measure two acceptance-rate diagnostics averaged across benchmarks. α_k is the probability that the drafter's greedy token at chain position k matches the target's greedy continuation, assuming positions $1, \ldots, k{-}1$ have all already matched, swept over $k = 1, \ldots, K{\cdot}M{=}8$ across two cross-block iterations. The chain position k does not correspond to a fixed block-internal index, since the boundary depends on how many positions of block 0 are taken before block 1 starts. $\alpha_{m,j}$ replots the rate at position j of block m under the same prior-match filter, with block 1 additionally assuming block 0 was fully accepted.

Figure 5(a) shows a smooth decay on both targets, with α_1 above 0.80 and α_8 around 0.37–0.54. Qwen3-8B drops sharper than Llama-3.1-8B, ending at $\alpha_8{=}0.369$ while Llama-3.1-8B retains 0.544. Figure 5(b) shows that $\alpha_{m,j}$ decays monotonically across the four positions within each block.



(a) Per-position α_k along the chain.

(b) Per-block per-position $\alpha_{m,j}$.

Figure 5: Acceptance rate diagnostics averaged across benchmarks. (a) Per-position α_k along the chain to depth $K{\cdot}M{=}8$, computed under the assumption that positions $1, \ldots, k-1$ all match the target's greedy continuation. (b) Per-block per-position $\alpha_{m,j}$ at position j of block m, computed under the assumption that positions $1, \ldots, j-1$ of the same block match; block 1 additionally assumes block 0 was fully accepted, since block 1 starts from the drafter's own cached $h^{(L)}$ at the last position of block 0.

Table 4: SpecBlock+adapt under single-GPU and dual-GPU deployment across three target models. Subscripts on each cell show the absolute gain over SpecBlock without adaptation. Spd is the speedup over vanilla decoding and τ is the average accepted length per verifier call.

		$T{=}0$				$T{=}1$			
		Spd		τ		Spd		τ	
Model	Bench	single	dual	single	dual	single	dual	single	dual
Llama-3.1-8B	MATH-500	$3.14_{+0.07}$	$3.12_{+0.05}$	$4.23_{+0.20}$	$4.20_{+0.17}$	$1.78_{+0.04}$	$1.77_{+0.03}$	$3.31_{+0.07}$	$3.26_{+0.02}$
	WMT-23	$2.81_{+0.02}$	$2.93_{+0.14}$	$3.81_{+0.06}$	$3.83_{+0.08}$	$2.36_{+0.05}$	$2.35_{+0.04}$	$3.57_{+0.16}$	$3.57_{+0.16}$
	Alpaca	$3.47_{+0.07}$	$3.46_{+0.06}$	$4.72_{+0.06}$	$4.69_{+0.03}$	$2.72_{+0.15}$	$2.66_{+0.09}$	$4.22_{+0.20}$	$4.19_{+0.17}$
	NQ	$3.51_{+0.51}$	$3.37_{+0.37}$	$5.41_{+1.01}$	$5.46_{+1.06}$	$2.39_{+0.47}$	$2.37_{+0.45}$	$4.57_{+0.97}$	$4.60_{+1.00}$
Qwen3-8B	MATH-500	$2.61_{+0.08}$	$2.59_{+0.06}$	$3.90_{+0.19}$	$3.89_{+0.18}$	$2.67_{+0.07}$	$2.64_{+0.04}$	$3.79_{+0.16}$	$3.70_{+0.07}$
	WMT-23	$2.34_{+0.04}$	$2.42_{+0.12}$	$3.36_{+0.08}$	$3.36_{+0.08}$	$2.19_{+0.08}$	$2.16_{+0.05}$	$3.31_{+0.12}$	$3.31_{+0.12}$
	Alpaca	$2.64_{+0.06}$	$2.63_{+0.05}$	$3.78_{+0.05}$	$3.77_{+0.04}$	$2.19_{+0.13}$	$2.15_{+0.09}$	$3.69_{+0.10}$	$3.66_{+0.07}$
	NQ	$2.69_{+0.43}$	$2.55_{+0.29}$	$3.75_{+0.54}$	$3.78_{+0.57}$	$2.27_{+0.46}$	$2.24_{+0.43}$	$3.55_{+0.49}$	$3.54_{+0.48}$
Qwen3-32B	MATH-500	$2.55_{+0.07}$	$2.53_{+0.05}$	$3.73_{+0.20}$	$3.71_{+0.18}$	$2.55_{+0.07}$	$2.51_{+0.03}$	$3.64_{+0.16}$	$3.56_{+0.08}$
	WMT-23	$2.24_{+0.04}$	$2.31_{+0.11}$	$3.22_{+0.07}$	$3.21_{+0.06}$	$2.14_{+0.07}$	$2.10_{+0.03}$	$3.22_{+0.12}$	$3.22_{+0.12}$
	Alpaca	$2.51_{+0.03}$	$2.52_{+0.04}$	$3.61_{+0.07}$	$3.56_{+0.02}$	$2.06_{+0.12}$	$2.02_{+0.08}$	$3.53_{+0.11}$	$3.48_{+0.06}$
	NQ	$2.57_{+0.40}$	$2.45_{+0.28}$	$3.61_{+0.54}$	$3.62_{+0.55}$	$2.19_{+0.44}$	$2.16_{+0.41}$	$3.40_{+0.48}$	$3.41_{+0.49}$

Position 1 of block 1 reaches 0.784 on Llama-3.1-8B and 0.748 on Qwen3-8B, both well above the last position of block 0 at 0.465 and 0.407, even though block 1 starts from the drafter's own cached state rather than the target's. The block boundary therefore acts as a recovery mechanism, supporting the block-iterative design over a single longer block of length $K{\cdot}M$.

D Adaptation deployment

D.1 Single-GPU and dual-GPU deployment

Cost-aware adaptation supports two deployment regimes: single-GPU, where the train stream runs on the same device as inference but on a separate CUDA stream, and dual-GPU, where the train stream runs on a second device and the two drafter copies sync via periodic in-place weight copy. Table 4 compares the two regimes on the three target models at $T{=}0$ and $T{=}1$, reporting raw Spd over vanilla decoding and accepted length τ for SpecBlock without adaptation, SpecBlock+adapt single-GPU, and SpecBlock+adapt dual-GPU.

The two regimes deliver comparable gains on every benchmark and across all three target models, with neither regime dominating. Single-GPU matches or even exceeds dual-GPU on Spd for MATH-500, Alpaca, and NQ across both temperatures, while dual-GPU wins on WMT-23 at $T{=}0$. The τ gap between the two regimes stays within 0.05 on most benchmarks, and both regimes recover most of the OOD acceptance loss on NQ. The dual-GPU regime additionally pays a cross-device

Table 5: SpecBlock+adapt on the 2K mixed stream after N epochs of adaptation over the stream. The first row reports SpecBlock without adaptation on the same stream as a no-adapt baseline; subscripts on each adapted row show the absolute gain over that baseline. Spd is the speedup over vanilla decoding and τ is the average accepted length per verifier call.

	Llama-3.1-8B				Qwen3-8B			
	Spd		τ		Spd		τ	
N	single	dual	single	dual	single	dual	single	dual
no adapt	3.05	3.05	4.19	4.19	2.41	2.41	3.47	3.47
1	$3.11_{+0.06}$	$3.11_{+0.06}$	$4.26_{+0.07}$	$4.28_{+0.09}$	$2.50_{+0.09}$	$2.51_{+0.10}$	$3.61_{+0.14}$	$3.63_{+0.16}$
2	$3.13_{+0.08}$	$3.13_{+0.08}$	$4.31_{+0.12}$	$4.32_{+0.13}$	$2.54_{+0.13}$	$2.55_{+0.14}$	$3.66_{+0.19}$	$3.68_{+0.21}$
4	$3.14_{+0.09}$	$3.15_{+0.10}$	$4.34_{+0.15}$	$4.36_{+0.17}$	$2.58_{+0.17}$	$2.58_{+0.17}$	$3.68_{+0.21}$	$3.72_{+0.25}$
6	$3.16_{+0.11}$	$3.16_{+0.11}$	$4.38_{+0.19}$	$4.41_{+0.22}$	$2.62_{+0.21}$	$2.63_{+0.22}$	$3.72_{+0.25}$	$3.74_{+0.27}$
8	$3.17_{+0.12}$	$3.17_{+0.12}$	$4.41_{+0.22}$	$4.45_{+0.26}$	$2.65_{+0.24}$	$2.65_{+0.24}$	$3.77_{+0.30}$	$3.76_{+0.29}$

weight sync latency, so frequent syncs do not automatically translate into higher throughput than the single-GPU regime, where the train stream shares the streaming multiprocessors with inference but avoids cross-device transfer. Cost-aware adaptation is therefore viable on a single-GPU deployment, and the dual-GPU variant is an option when an extra device is available for the train stream.

D.2 Mixed-task adaptation

Production traffic is rarely a single homogeneous benchmark: user requests typically mix tasks such as math, translation, instruction following, and QA, and may revisit similar prompt patterns over time as the same users return. To inspect adaptation behavior under such heterogeneous and repeated traffic, we construct a mixed stream by sampling equal proportions of prompts from MATH-500, WMT-23, Alpaca, and NQ, with a total stream size of 2K queries. We then sweep the number of full passes over the mixed stream $N \in \{1, 2, 4, 6, 8\}$ to mimic the steady-state regime where the drafter has seen the user's task distribution multiple times.

Both Spd and τ grow monotonically with the number of adapt epochs N on both target models and both deployment regimes. After $N{=}8$ epochs, the adaptation lifts τ by 0.22–0.30 over the no-adapt baseline and Spd by roughly 3.9–10.0%, with the larger gains on Qwen3-8B where the original drafter has more headroom. Dual-GPU is consistently a small step ahead of single-GPU but the gap stays under 0.05 in τ and under 1% in Spd, again echoing that cross-device sync latency offsets some of the benefit of separating the train stream from inference. The monotonic growth confirms that cost-aware adaptation can extract additional accepted length when the same mixed traffic is revisited, mirroring a deployment scenario where returning users supply repeated task patterns to the same drafter instance.

E Case Study

To illustrate the structural difference between SpecBlock and EAGLE-3 trees and how each translates into accepted tokens, we run both drafters on the same prompt with the Qwen3-8B target under greedy decoding. Figure 6 shows the first 30 committed tokens of each verifier-committed response, with green shading for tokens drafted and accepted by the verifier and red shading for bonus tokens sampled from the target after acceptance. The two drafters commit nearly the same content, and most of the divergence between them is in which positions land as bonuses rather than as drafted accepts.

Figure 7 visualizes the iter-0 draft tree of each drafter on the same prompt; node coloring shows which drafter forward produced each token. EAGLE-3 grows depth-by-depth, so reaching depth 7 costs seven sequential drafter forwards, with each forward shown in a different color, fwd 1 through fwd 7. SpecBlock pays only two forwards on the same prompt: block-1 emits all $K{=}4$ chain positions in one forward drawn horizontally, and block-2 then batches additional chains from rank-head–selected starts; the two blocks are colored differently. Both drafters land on the same accepted prefix `F ib onacci Sequence`. EAGLE-3 keeps drafting and the verifier walks three more depths, committing eight tokens including a bonus over seven forwards. SpecBlock stops at the block-1

PROMPT

```
Write a Python function to compute the Fibonacci sequence.
```

EAGLE-3 RESPONSE

```
**Fibonacci Sequence Function**
==================================

The Fibonacci sequence is a series of numbers where a number is the sum of
the two preceding ones, usually starting with 0 and 1.

### Recursive Implementation
```

```
"'python
def fibonacci_recursive(n):
_ """
_ Compute the ...
```

SPECBLOCK RESPONSE

```
Fibonacci Sequence Function**
==================================

The Fibonacci sequence is a series of numbers where a number is the sum of
the two preceding ones, usually starting with 0 and 1.

### Recursive Implementation
```

```
"'python
def fibonacci_recursive(n):
_ """
_ Compute the nth ...
```

Figure 6: Case study response with per-token acceptance shading. green token indicates a token drafted by the drafter and accepted by the verifier; red token indicates a bonus token sampled from the target after acceptance. Each response is shown for the first 30 committed tokens, with a trailing . . . marking the omitted tail.

chain and commits five tokens including a bonus over two forwards. The case makes the SpecBlock trade-off visible: SpecBlock trades a slightly shorter accepted run per iteration for an iteration that finishes in two drafter calls instead of seven.

(a) EAGLE-3

```
Write a Python function to compute the Fibonacci sequence. <|eot|> \ n\ n

root
└─ [F] ✓
    ├─ [ib] ✓
    |   ├─ [onacci] ✓
    |   |   ├─ [ Sequence] ✓
    |   |   |   ├─ [ Function] ✓
    |   |   |   |   ├─ [**\ n] ✓
    |   |   |   |   |   ├─ [================] ✓
    |   |   |   |   |   └─ [The]
    |   |   |   |   ├─ [**\ n\ n]
    |   |   |   |   ├─ [ to]
    |   |   |   |   └─ [:]
    |   |   |   ├─ [ Calculator]
    |   |   |   ├─ [ Generator]
    |   |   |   ├─ [ with]
    |   |   |   └─ ... (9 more siblings)
    |   |   ├─ [Sequence]
    |   |   ├─ [ Series]
    |   |   └─ [ Function]
    |   └─ [()]
    └─ [(n]
        └─ [()]
```

(b) SpecBlock

```
Write a Python function to compute the Fibonacci sequence. <|eot|> \ n\ n

root
├─ [F] ✓ — [ib] ✓ — [onacci] ✓ — [ Sequence] ✓
|   |       |       |             |
|   |       |       |             ├─ [ using]
|   |       |       |             |   ├─ [ Python]
|   |       |       |             |   └─ [The]
|   |       |       |             ├─ [ in]
|   |       |       |             ├─ [**\ n]
|   |       |       |             └─ ... (8 more siblings)
|   |       |       ├─ [ Sequence]
|   |       |       ├─ [ using]
|   |       |       └─ ... (4 more siblings)
|   |       ├─ [ Fibonacci] — [onacci] — [ Sequence]
|   |       ├─ [:**]
|   |       ├─ [ Implementation] — [ of]
|   |       └─ ... (9 more siblings)
|   └─ ...
├─ [ Fibonacci] — [onacci] — [ Sequence]
├─ [The] — [ Fibonacci] — [ Sequence]
├─ [What] — [ is] — [ the] — [ Fibonacci] — [ Sequence]
├─ [Python]
└─ [Overview]
```

Figure 7: Per-iteration draft tree for the prompt `Write a Python function to compute the Fibonacci sequence.` EAGLE-3 grows depth-by-depth at one drafter forward per depth; each of the seven forwards is shown in a different color, fwd 1 through fwd 7. SpecBlock reaches a comparable accepted prefix in only two forwards, with block-1 and block-2 shown in two different colors. Tokens marked ✓ are on the verifier-walked accept path; "N more siblings" counts candidates omitted for clarity.

F Comparison with Recent Drafters

We compare SpecBlock with DFlash [47] and DSpark [48] on Llama-3.1-8B-Instruct using a single A100-80GB GPU at batch size 1. DFlash and DSpark use the public DeepSpec checkpoints and evaluation code with block7, proposing seven tokens in a linear sequence. EAGLE-3 and SpecBlock use their respective tuned tree configurations.

Table 6: Throughput (tokens/s) on Llama-3.1-8B-Instruct at batch size 1. Parentheses show speedup over target-only decoding.

Dataset	Target-only	EAGLE-3	SpecBlock	DFlash	DSpark
Temperature = 0					
HumanEval	36.87	132.33 (3.59×)	**144.53 (3.92×)**	84.00 (2.28×)	110.96 (3.01×)
MATH-500	36.72	97.68 (2.66×)	112.73 (3.07×)	85.11 (2.32×)	**117.82 (3.21×)**
Alpaca	35.77	119.83 (3.35×)	**121.62 (3.40×)**	77.74 (2.17×)	104.68 (2.93×)
NQ	29.01	70.21 (2.42×)	87.04 (3.00×)	**122.20 (4.21×)**	111.59 (3.85×)
MT-Bench	36.23	116.68 (3.22×)	112.31 (3.10×)	**134.98 (3.73×)**	110.58 (3.05×)
WMT-23	35.71	81.41 (2.28×)	99.62 (2.79×)	**129.99 (3.64×)**	102.89 (2.88×)
Temperature = 1					
HumanEval	36.08	107.52 (2.98×)	**115.82 (3.21×)**	56.21 (1.56×)	61.02 (1.69×)
MATH-500	35.86	47.34 (1.32×)	62.40 (1.74×)	35.37 (0.99×)	**71.67 (2.00×)**
Alpaca	35.90	84.72 (2.36×)	**92.26 (2.57×)**	37.57 (1.05×)	75.37 (2.10×)
NQ	34.28	56.90 (1.66×)	65.82 (1.92×)	55.04 (1.61×)	**67.22 (1.96×)**
MT-Bench	36.07	53.38 (1.48×)	54.11 (1.50×)	61.32 (1.70×)	**71.57 (1.98×)**
WMT-23	34.34	62.16 (1.81×)	**79.33 (2.31×)**	55.36 (1.61×)	53.04 (1.55×)

Table 6 shows that SpecBlock achieves the highest mean throughput across the six tasks at both temperatures: 112.98 tokens/s at $T{=}0$ and 78.29 tokens/s at $T{=}1$. DFlash leads on three tasks at $T{=}0$, while DSpark leads on MATH-500 at both temperatures and on NQ and MT-Bench at $T{=}1$.

G Effect of Batch Size

We evaluate fixed-batch HuggingFace inference on Llama-3.1-8B-Instruct using a single A100-80GB GPU at batch sizes $B \in \{1, 4, 8, 16, 32\}$ and temperatures $T \in \{0, 1\}$. Table 7 reports arithmetic means across HumanEval, MATH-500, Alpaca, NQ, MT-Bench, and WMT-23. Speedup is computed against target-only decoding on each task before averaging.

Table 7: Fixed-batch HuggingFace performance on Llama-3.1-8B-Instruct, averaged across six tasks. Throughput is in tokens/s; Spd is speedup over target-only decoding, τ is accepted length, and $T_{\mathcal{D}}\%$ is the drafting-time share.

		Target-only	EAGLE-3				SpecBlock			
T	B	Throughput	Throughput	Spd	τ	$T_{\mathcal{D}}\%$	Throughput	Spd	τ	$T_{\mathcal{D}}\%$
0	1	35.05	103.02	2.92	5.89	31.0	112.98	3.21	4.41	16.0
0	4	70.52	139.39	1.989	5.00	43.0	182.40	2.587	3.25	25.1
0	8	118.09	169.28	1.454	4.99	50.9	203.26	1.740	3.23	27.4
0	16	142.20	142.84	1.027	5.97	60.0	235.35	1.679	4.23	27.9
0	32	163.79	157.05	0.963	5.99	62.3	265.28	1.625	4.26	26.5
1	1	35.42	68.67	1.94	4.67	30.0	78.29	2.20	3.73	14.0
1	4	71.09	84.09	1.159	2.46	37.5	68.19	0.953	1.86	13.9
1	8	121.51	100.19	0.817	2.45	44.9	78.65	0.651	1.96	15.1
1	16	158.24	91.38	0.573	3.63	54.8	100.59	0.630	2.97	17.1
1	32	172.66	87.64	0.514	3.44	52.8	123.51	0.734	2.98	18.2

At $T{=}0$, SpecBlock achieves higher mean throughput than EAGLE-3 at every batch size despite its shorter accepted length. At $B{=}32$, it retains a mean speedup of $1.63\times$ over target-only decoding, compared with $0.96\times$ for EAGLE-3. At $T{=}1$, SpecBlock leads EAGLE-3 at $B{=}1, 16, 32$ but falls behind at $B{=}4, 8$; both methods have mean speedups below one at $B \geq 8$. The lower acceptance under sampling makes verification and sampling overhead more prominent, while EAGLE-3's sequential drafting becomes increasingly costly at larger batches.

H Effect of Tree Branching

To isolate the benefit of block drafting without tree branching, we restrict SpecBlock and EAGLE-3 to a single path of eight candidate tokens per iteration on Llama-3.1-8B-Instruct. SpecBlock generates two blocks of four tokens, while EAGLE-3 generates the eight tokens sequentially. Table 8 reports throughput, speedup, and accepted length at $T{=}0$ and $T{=}1$.

Table 8: Linear-path decoding on Llama-3.1-8B-Instruct with eight candidate tokens per iteration and no tree branching. Throughput is in tokens/s; Spd is speedup over target-only decoding, and τ is accepted length. Mean denotes the arithmetic mean across tasks.

	Target-only	EAGLE-3			SpecBlock		
Dataset	Throughput	Throughput	Spd	τ	Throughput	Spd	τ
Temperature = 0							
HumanEval	36.90	69.05	1.87	5.10	66.48	1.80	3.27
MATH-500	36.38	49.60	1.36	4.09	50.10	1.38	2.41
Alpaca	29.58	47.56	1.61	4.17	50.90	1.72	2.77
NQ	29.01	38.43	1.32	3.98	55.23	1.90	2.87
MT-Bench	35.41	51.55	1.46	4.23	49.54	1.40	2.73
WMT-23	35.71	37.83	1.06	3.09	43.39	1.22	2.44
Mean	33.83	49.00	1.45	4.11	52.61	1.57	2.75
Temperature = 1							
HumanEval	35.77	34.24	0.96	3.80	52.74	1.48	2.84
MATH-500	32.95	24.35	0.74	2.20	31.05	0.94	1.60
Alpaca	33.90	28.29	0.83	2.77	41.26	1.22	2.37
NQ	34.28	26.75	0.78	2.43	35.28	1.03	2.31
MT-Bench	32.81	28.34	0.86	2.49	37.52	1.14	2.11
WMT-23	34.34	28.12	0.82	2.49	35.86	1.04	1.97
Mean	34.01	28.35	0.83	2.70	38.95	1.14	2.20

Without branching, SpecBlock retains mean speedups of $1.57\times$ at $T{=}0$ and $1.14\times$ at $T{=}1$, compared with $1.45\times$ and $0.83\times$ for EAGLE-3. Its shorter accepted paths are offset by fewer sequential drafter forwards, showing that block drafting provides a throughput benefit even without the additional candidate coverage of a tree.

I Evaluation in SGLang

We evaluate an initial SGLang implementation at batch size 1 across the six tasks. SpecBlock achieves mean throughputs of 210.38 and 185.95 tokens/s at $T{=}0$ and $T{=}1$, respectively, yielding $2.37\times$ and $2.12\times$ speedups over target-only decoding. EAGLE-3 remains faster at 231.64 and 194.93 tokens/s. Profiling identifies remaining overhead in rank-guided tree construction, tree verification, and accepted-path state updates, with further work needed on CUDA Graph integration.

J Evaluation on MetaX C500

We evaluate greedy decoding on a single MetaX C500 GPU at batch size 1 using Llama-3.1-8B-Instruct and Qwen3-8B, with thinking disabled for Qwen3. We use 164 HumanEval prompts, 80 MT-Bench dialogues, and 200 examples each from MATH-500, Alpaca, NQ-Open, and WMT-23, with generation capped at 1,024 new tokens per request. EAGLE-3 and SpecBlock use their respective configurations with 60 and 90 draft descendants, excluding the root. Throughput is generated tokens divided by wall-clock runtime, and speedup is computed against target-only decoding on the same task. Accepted length is the total number of accepted draft tokens divided by the number of verification rounds, plus one bonus token.

Table 9: Greedy decoding on a single MetaX C500 at batch size 1. Throughput is in tokens/s; Spd is speedup over target-only decoding, and τ is accepted length including the bonus token. Mean denotes the arithmetic mean across tasks.

	Target-only	EAGLE-3			SpecBlock		
Dataset	Throughput	Throughput	Spd	τ	Throughput	Spd	τ
Llama-3.1-8B-Instruct							
HumanEval	36.75	102.75	2.80	6.98	119.80	3.26	5.08
MATH-500	36.79	88.28	2.40	5.95	90.66	2.46	3.87
Alpaca	36.59	93.86	2.57	6.43	109.65	3.00	4.56
NQ-Open	36.51	74.75	2.05	5.16	91.42	2.50	3.77
MT-Bench	36.74	90.36	2.46	6.22	97.75	2.66	4.38
WMT-23	35.94	60.21	1.68	4.24	84.66	2.36	3.61
Mean	36.55	85.03	2.32	5.83	98.99	2.71	4.21
Qwen3-8B							
HumanEval	26.90	73.75	2.74	4.83	87.32	3.25	4.87
MATH-500	27.03	71.79	2.66	4.69	86.32	3.19	4.81
Alpaca	26.23	71.72	2.73	4.67	80.41	3.07	4.43
NQ-Open	26.65	62.80	2.36	4.07	74.01	2.78	3.85
MT-Bench	26.50	66.68	2.52	4.49	59.39	2.24	3.65
WMT-23	25.72	45.79	1.78	3.07	61.79	2.40	3.40
Mean	26.50	65.42	2.46	4.30	74.87	2.82	4.17

Table 9 shows that SpecBlock improves mean throughput over EAGLE-3 by 16.4% on Llama-3.1-8B-Instruct and 14.4% on Qwen3-8B. It is faster on all six Llama tasks and five of the six Qwen tasks, with MT-Bench as the exception. Table 10 breaks down iteration time: SpecBlock reduces drafting time from 18.71 to 9.18 ms on Llama and from 18.57 to 11.43 ms on Qwen, reductions of 50.9% and 38.4%, respectively.

Table 10: Stage time on MetaX C500 at $T{=}0$, in ms per iteration, with the fraction of iteration time in parentheses. Times are pooled across all six tasks by dividing total stage time by total iterations.

Target	Method	Draft	Target	Verify
Llama-3.1-8B	Target-only	0.00 (0.0%)	27.28 (100.0%)	0.00 (0.0%)
Llama-3.1-8B	EAGLE-3	18.71 (28.9%)	40.54 (62.7%)	5.43 (8.4%)
Llama-3.1-8B	SpecBlock	9.18 (22.1%)	29.65 (71.3%)	2.76 (6.6%)
Qwen3-8B	Target-only	0.00 (0.0%)	37.46 (100.0%)	0.00 (0.0%)
Qwen3-8B	EAGLE-3	18.57 (29.8%)	37.61 (60.4%)	6.05 (9.7%)
Qwen3-8B	SpecBlock	11.43 (20.7%)	39.75 (71.9%)	4.11 (7.4%)

K Limitations

SpecBlock builds its verifier tree using the rank head's per-position prediction, which decides how many sibling alternatives each position carries and whether the position starts a later block. The shape of the tree therefore depends on how accurate this prediction is. The four-bucket classifier already does better than a uniform tree of the same node budget, as the rank-head ablation in Table 2 confirms, but it is not perfectly accurate. On a non-trivial fraction of positions the prediction misses by one or two buckets, so the position ends up with too few siblings when the target token sits far down the drafter's distribution, and too many when the drafter is already confident. A finer-grained or more accurate rank head would let SpecBlock spend its verifier budget more tightly.

The block width $K{=}4$ is decided at training time and cannot be changed at inference, because the layer-wise shift mechanism is built around a specific K. The number of iterative blocks M, in contrast, can be extended naturally at inference. A drafter trained with $M{=}3$ continues to work at $M{=}4$, since each additional block simply reuses the same drafter forward on a new starting point. In our experiments we use $M{=}2$ across deployments rather than searching M per workload. Both K at training and M at inference interact with the cost ratio between the target and the drafter. A much larger target makes each verifier call expensive and rewards a larger K or deeper block stacking. When the two costs are closer, smaller values save drafter time without losing much. Workloads whose acceptance distribution differs noticeably from our training mix may also prefer values we did not select.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction claim four contributions: the block-iterative drafter that produces K dependent positions per forward, the layer-wise shift that preserves within-block dependence, the co-trained rank head that shapes the verifier tree, and the cost-aware bandit that refreshes the drafter at serving time. Section 4 backs the speedup claim of 8 to 13% mean over EAGLE-3 across three target models, with cost-aware adaptation extending the gain to 11 to 19%. Table 1 reports the per-benchmark breakdown.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss SpecBlock's limitations in Appendix K. The rank head's four-bucket prediction is not perfectly accurate, which leaves some verifier budget mis-allocated across positions. And the block width K is decided at training time and cannot be changed at inference.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4 states the datasets, target models, training hyperparameters, architectural choices, and hardware setup. Appendix A gives the remaining implementation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is available at https://github.com/shiweijiezero/SpecBlock with training and evaluation scripts. The UltraChat-200K and ShareGPT training data and the evaluation benchmarks are publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 specifies the architectural hyperparameters, the training optimizer and schedule, the datasets, and the inference setup. Appendix A adds the remaining details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Each τ and speedup entry in Table 1 averages over hundreds of verifier calls per benchmark.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4 reports the inference setup of a single A100-80GB at batch size 1, and Appendix A reports the training cost of roughly 3,000 A100-80GB GPU-hours per drafter.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The work conforms to the NeurIPS Code of Ethics. SpecBlock is an inference-acceleration method that uses publicly available datasets and pre-trained models, and does not involve human subjects.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: SpecBlock lowers the compute and energy cost of LLM inference without introducing new capabilities. Beyond the direct cost saving, the work also highlights that drafter compute itself, not only acceptance length, should be a primary optimization target in speculative decoding, which we hope encourages more energy-efficient serving research. Negative impacts are inherited from the underlying LLMs the method accelerates.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: SpecBlock drafters are inference accelerators for existing LLMs and do not pose elevated misuse risk. The paper does not release new pretrained models or datasets.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Section 4 cites the target models, training datasets, and evaluation benchmarks used. All are publicly available and used under their original licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The code repository at `https://github.com/shiweijiezero/SpecBlock` provides a README with training and evaluation instructions.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We used LLMs to help us understand prior work and polish the writing. The target LLMs accelerated in our experiments, Llama-3.1-8B, Qwen3-8B, and Qwen3-32B, are described in Section 4.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.