SPIRIT: Speed-Driven Online Adaptation for Self-Speculative Decoding

Chenghao Liu Chengbo Zhan Hui Lu Jia Rao
Department of Computer Science and Engineering
The University of Texas at Arlington
Arlington, TX 76019, USA
{cxl5192,cxz3901}@mavs.uta.edu
{hui.lu,jia.rao}@uta.edu

Abstract

Self-speculative decoding offers a plug-and-play route to accelerating autoregressive large language model (LLM) inference, but its practical success hinges on identifying effective layer-skipping configurations online without retraining or offline profiling. Existing methods mainly optimize agreement-based proxies that capture draft–target consistency, yet overlook intrinsic runtime differences across draft configurations. Consequently, configurations with similar acceptance can still yield substantially different realized decode throughput. We present SPIRIT, a speed-driven online adaptation framework for self-speculative decoding that directly optimizes realized decode throughput—measured as generated tokens per second—using window-level runtime feedback. Our key insight is that speed is not determined by agreement alone: it is jointly shaped by draft–target consistency and the intrinsic speed advantage of the draft configuration. Empirically, the skip ratio serves as a first-order control variable, capturing the dominant consistency–speed trade-off while sharply shrinking the search space over layer-skipping patterns. SPIRIT therefore first identifies an effective skip ratio and then refines the layer-skipping pattern within it. Combined with length-aware normalization and shift-aware profile-based adaptation, this design enables rapid adaptation to task shifts in non-stationary request streams. Across diverse tasks, model scales, and serving regimes, SPIRIT consistently outperforms strong training-free baselines and achieves up to $1.74\times$ speedup with FP16 self-drafts and $2.48\times$ with NF4 self-drafts while preserving the target model's output distribution, establishing a practical foundation for plug-and-play, high-throughput LLM inference.

1 Introduction

Large language models (LLMs) are increasingly deployed in practical systems, yet their efficiency remains bottlenecked by autoregressive decoding, where tokens are generated sequentially and each step requires a full model invocation [1; 2; 3]. As models grow and outputs become longer, maintaining efficient decoding becomes difficult [4; 5; 6]. In deployment, the most practical acceleration methods are those that improve efficiency without requiring retraining, auxiliary draft models, or costly model-specific profiling [7; 8]. Speculative decoding is a promising route toward lossless acceleration: a fast draft mechanism proposes multiple candidate tokens and the target model verifies them in parallel, reducing decode cost while preserving the target model's output distribution [9; 10; 11].

Speculative decoding methods differ not only in speedup, but also in the deployment assumptions imposed on the draft path [12; 13; 14]. Trained-drafter methods such as EAGLE and Medusa achieve strong speedups [15; 16], but require target-specific heads to be trained and deployed: EAGLE reports 0.99B trainable parameters and 1–2 days of training for its 70B-scale autoregression head [17], while

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

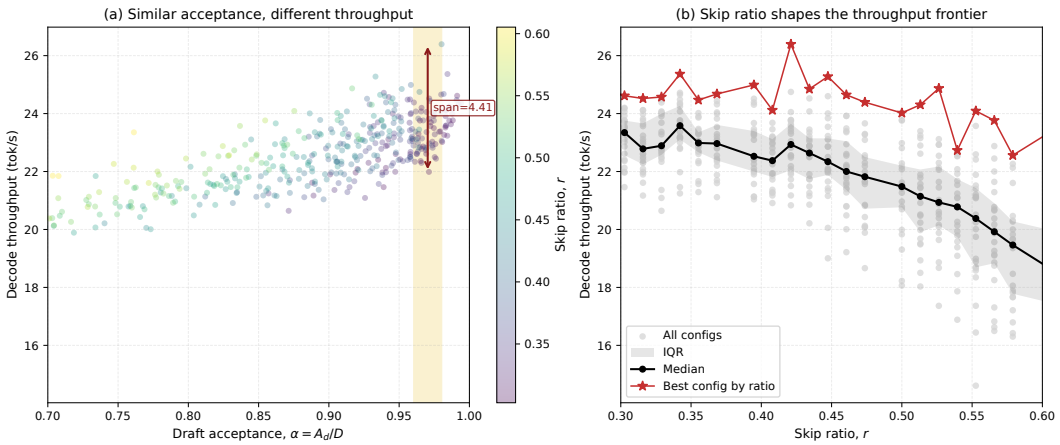


Figure 1: **Why agreement alone is not the objective.** (a) Similar draft acceptance can still yield different decode throughput. (b) Skip ratio shapes the dominant throughput frontier.

Medusa trains multiple decoding heads [16]. Trained self-speculative methods reduce reliance on a separate drafter, but still modify the model or add trainable components: LayerSkip uses layer-dropout and early-exit training [18], while Kangaroo trains an adapter on top of a shallow sub-network [19]. Offline self-speculative methods such as Draft & Verify avoid training, but use costly pre-inference Bayesian optimization [20] to obtain a fixed skipping configuration, with search cost reported up to 20 hours for LLaMA-2-70B [21; 22]. SPIRIT targets the complementary online self-speculative regime, inducing the draft path by layer skipping without auxiliary modules, target-specific training, or offline calibration. Recent training-free methods such as SWIFT [22] and CLaSp [23] move beyond fixed skipping configurations by adapting skipped layers online or in context, reflecting request- and task-dependent skipping behavior. SWIFT is the closest strong open-source baseline in this regime, but existing adaptive controllers still rely on agreement-oriented proxies—token matchness in SWIFT and hidden-state similarity in CLaSp—rather than realized decode throughput [22; 23].

Our key observation is simple but important: *agreement is a useful signal, but not the optimization objective*. In self-speculative decoding, realized decode throughput depends on both draft acceptance and the intrinsic execution rate of the draft configuration. Consequently, configurations with similar agreement can still yield substantially different wall-clock throughput. This mismatch becomes decisive in online control: the controller must optimize realized decode throughput under the current workload, rather than draft–target alignment alone. Figure 1(a) makes this concrete: configurations with similar draft acceptance can yield markedly different throughput, so agreement alone cannot identify the throughput-optimal configuration. Figure 1(b) reveals the structure SPIRIT exploits: the skip ratio organizes the dominant throughput frontier by setting the coarse consistency–speed trade-off, while within-ratio spread leaves room for layer-pattern refinement. SPIRIT therefore adapts coarse-to-fine: it first localizes a high-throughput skip-ratio regime, then refines the skipped-layer pattern within the reduced search space.

Motivated by this perspective, we present **SPIRIT**, a throughput-aligned online adaptation framework for self-speculative decoding, with continuous runtime monitoring and shift-aware profile-based adaptation across non-stationary request streams. Our contributions are threefold. **(i)** We formulate training-free self-speculative decoding as a *throughput-aligned* online optimization problem that directly maximizes realized decode throughput, rather than relying on agreement-oriented surrogates. This objective captures both draft–target consistency and intrinsic draft speed. **(ii)** We develop a robust online feedback mechanism based on *window-level* throughput statistics with *length-aware normalization*, enabling rapid adaptation from fine-grained runtime signals while correcting for sequence-length bias introduced by KV-cache growth during generation. **(iii)** We introduce a hierarchical adaptation framework that combines *ratio-first* two-stage search within requests and *shift-aware profile-based* cross-request adaptation under workload shifts, enabling efficient configuration search and profile reuse across non-stationary request streams.

We evaluate SPIRIT across diverse tasks, model scales, and both stationary and mixed-stream settings. SPIRIT achieves up to 1.68× speedup on LLaMA-2-13B, up to 1.74× on LLaMA-2-70B [24], and

up to 1.60× under repeated task switching. These results demonstrate that speed-driven adaptation remains effective at larger model scales and under non-stationary workloads, providing a practical path toward plug-and-play, high-throughput LLM inference.

2 Background and Motivation

2.1 Layer-Skipping Self-Speculation

Self-speculative decoding constructs a draft path within the target model itself, avoiding an additional draft model and its associated memory overhead, while using the full target model to verify and commit tokens. Within this family, layer skipping provides a particularly practical training-free draft mechanism: Draft & Verify first showed that bypassing intermediate transformer layers [25] can form an effective self-draft path without training, but its skipped-layer set is selected by offline Bayesian optimization and then fixed during serving [21]. SWIFT further showed that layer sparsity is task-specific and introduced on-the-fly skipped-layer adaptation [22], while CLaSp adjusts skipped layers using in-context hidden-state feedback [23]. Together, these results establish layer skipping as a viable training-free self-speculative mechanism, but also show that useful skipping behavior is workload-dependent rather than universal.

At each speculative step, the draft path proposes up to γ tokens, which the verifier checks in parallel. If $a_t \in \{0, \ldots, \gamma\}$ draft tokens are accepted at step t, the step commits

$$g_t = a_t + 1 \tag{1}$$

output tokens: a_t accepted draft tokens plus one additional verifier-produced token. Thus, online control should reason about *committed output tokens*, rather than accepted draft tokens alone. We describe the greedy setting [26]; the same notation extends to standard lossless verification under sampling.

2.2 From Layer-Skipping Masks to Structured Control

A raw layer-skipping mask over transformer sublayers is combinatorial and conflates two decisions with different runtime effects: how much computation is removed and where it is removed [27]. We therefore represent a layer-skipping draft configuration as

$$c = (r, L), \tag{2}$$

where r is the skip ratio and $L \subseteq \{\mathrm{Attn}_\ell, \mathrm{MLP}_\ell\}_{\ell=1}^{N}$ denotes the skipped Attention/MLP sublayers. In our search space, the skip budget induced by r is $|L| = \lfloor 2Nr \rfloor$.

The skip ratio r controls the coarse aggressiveness of the draft path: increasing r skips more sublayers and generally reduces draft latency, but may weaken agreement with the verifier. The pattern L refines the execution path within the same skip budget: different placements of skipped Attention/MLP sublayers preserve different information flows and incur different runtime costs [28; 29]. This factorization separates *how much* to skip from *which sublayers* to skip, motivating SPIRIT's ratio-first localization followed by layer-pattern refinement under a throughput-aligned objective.

3 Method

3.1 Execution-Coupled Runtime Adaptation

SPIRIT casts training-free self-speculative decoding as a closed-loop runtime control problem over structured layer-skipping configurations $c = (r, L)$, where r is the skip ratio and L is the concrete skip-layer pattern. Rather than separating configuration search from serving, SPIRIT interleaves adaptation directly with speculative decoding. Each runtime window plays a dual role: it advances decoding while exposing execution statistics of the active configuration. These statistics are converted into a throughput-aligned control signal for selecting the next configuration at window boundaries.

This execution-coupled design induces two adaptation timescales, with feedback obtained directly from ordinary speculative execution. At runtime-window boundaries, SPIRIT applies lightweight controller updates using statistics emitted by the preceding window, without any auxiliary validation

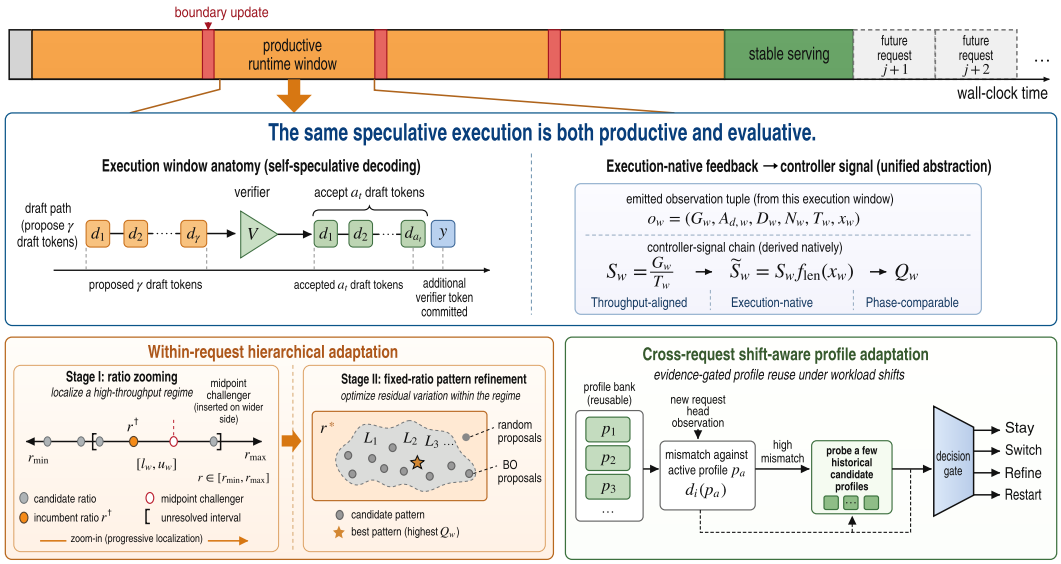


Figure 2: **SPIRIT as closed-loop runtime control for self-speculative decoding.** Each runtime window is both productive and evaluative: ordinary speculative execution advances decoding while emitting the statistics used for lightweight boundary updates (red), without any auxiliary validation stage. This induces two nested control loops: within-request hierarchical adaptation from ratio search to fixed-ratio layer refinement, and cross-request shift-aware profile reuse via stay, switch, refine, or restart.

stage or extra model forward pass. Within a request, each runtime window is both productive and evaluative: it advances decoding while supplying the feedback needed for coarse-to-fine search, first over skip ratios and then over layer-skipping patterns. Across requests, SPIRIT monitors request-head behavior and decides whether to keep the active profile, switch to a historical profile, perform short local refinement, or restart online search.

3.2 Execution-Native Feedback and Boundary Control

SPIRIT uses ordinary speculative execution itself as the feedback channel for online adaptation, rather than relying on an auxiliary validation or similarity-scoring stage. For a runtime window w executed under the active configuration c_w, the controller receives the observation

$$o_w = \big(G_w, A_{d,w}, D_w, N_w, T_w, x_w\big), \tag{3}$$

where the tuple records committed output tokens, accepted draft tokens, drafted tokens, verification rounds, elapsed decode time, and a summary of sequence-length context. As illustrated by the short update phases in Figure 2, these statistics are emitted directly by the standard draft–verify process and exposed at runtime-window boundaries.

Given controller state z_w and window observation o_w, SPIRIT applies a boundary update

$$z_{w+1} = \mathcal{U}(z_w, o_w), \qquad c_{w+1} = \pi(z_{w+1}), \tag{4}$$

where $\mathcal{U}$ updates the controller state and π selects the next draft configuration to evaluate. Crucially, the boundary update is both immediate and execution-native: it is applied at the runtime-window boundary and depends only on statistics emitted by the preceding window. It therefore incurs only lightweight controller computation and no additional model forward pass.

The resulting interface makes each runtime window both *productive* and *evaluative*: while evaluating the active configuration, it continues committing output tokens and exposing the feedback needed for online control. Boundary updates, by contrast, are non-productive but lightweight, serving only to revise the controller state and dispatch the next configuration. SPIRIT therefore adapts online without interrupting speculative decoding with a separate validation or similarity-scoring stage.

3.3 Throughput-Aligned Controller Signal

A runtime controller for self-speculative decoding should not be evaluated by how well it diagnoses draft–target agreement in isolation, but by how effectively it steers the system toward higher realized output production. The resulting control signal must therefore satisfy three properties: it must be *utility-aligned*, because deployment ultimately cares about wall-clock efficiency; *execution-native*, because online adaptation should be driven by ordinary speculative decoding rather than auxiliary scoring stages; and *phase-comparable*, because decoding speed naturally drifts as sequence length grows. SPIRIT instantiates this perspective by turning execution-native observations into a throughput-aligned controller signal measured over runtime windows.

Concretely, once the controller is required to optimize realized serving utility, the natural quantity to score is window-level committed-token throughput. For a runtime window w, we define the raw throughput score as

$$S_w = \frac{G_w}{T_w}, \tag{5}$$

where G_w is the number of committed output tokens produced in the window and T_w is the corresponding decode time, excluding prefill. By construction, each speculative verification round contributes one committed verifier token in addition to any accepted draft tokens, so

$$G_w = A_{d,w} + N_w. \tag{6}$$

Substituting this relation into Eq. (5) gives

$$S_w = \frac{A_{d,w} + N_w}{T_w} = \left(\frac{A_{d,w}}{D_w} + \frac{N_w}{D_w} \right) \frac{D_w}{T_w} = (\alpha_w + \beta_w)\nu_w, \qquad D_w > 0, \tag{7}$$

where $\alpha_w = A_{d,w}/D_w$, $\beta_w = N_w/D_w$, and $\nu_w = D_w/T_w$ denote draft acceptance, the verifier-token bonus per drafted token, and the intrinsic execution rate of the draft configuration, respectively.

This decomposition makes the limitation of agreement-only control explicit. Realized throughput is jointly governed by an *acceptance-side gain*, $(\alpha_w + \beta_w)$, and an *execution-side gain*, ν_w. The former determines how effectively drafted computation is converted into committed output, while the latter determines how quickly that drafted computation is carried out. Thus, similar draft–target consistency does not imply similar throughput: once execution rates differ, the resulting wall-clock efficiency can diverge substantially. SPIRIT therefore treats realized committed-token throughput, rather than an agreement-oriented proxy, as the controller objective.

A further challenge is that raw throughput is inherently nonstationary across decoding phases. Even under a fixed configuration, decode speed systematically drifts as generation proceeds: longer prefixes enlarge the KV cache, increase memory traffic, and thereby reduce token production speed. A controller that compares raw throughput across windows would therefore conflate configuration quality with phase-dependent slowdown. SPIRIT addresses this issue through length-aware normalization, which debiases and partially stationarizes the controller signal across decoding phases. Specifically, it transforms the raw throughput score as

$$\tilde{S}_w = S_w \cdot f_{\mathrm{len}}(x_w), \tag{8}$$

where x_w denotes the average prefix length of window w, and $f_{\mathrm{len}}(\cdot)$ is a correction factor estimated online from streaming window statistics. This normalization factors out the dominant predictable drift induced by prefix growth, so that variations in $\tilde{S}_w$ more faithfully reflect configuration quality rather than phase-dependent runtime effects; implementation details are deferred to Appendix B.1. Moreover, to suppress short-window noise while preserving online responsiveness, SPIRIT applies an exponential moving average to the normalized throughput score to obtain the controller-facing signal:

$$Q_w = \lambda \tilde{S}_w + (1 - \lambda) Q_{w-1}, \tag{9}$$

where $\lambda \in (0, 1]$ controls the degree of smoothing. The resulting score Q_w is directly throughput-aligned, execution-native, and phase-comparable, and it is this signal that drives the hierarchical adaptation procedure described next.

3.4 Within-Request Hierarchical Adaptation

SPIRIT exploits a key structure in the layer-skipping space: the skip ratio determines the dominant throughput regime, while the concrete skip-layer pattern explains residual variation within that regime. This structure naturally induces a hierarchical within-request controller in which Stage I first localizes a high-performing ratio regime and Stage II then refines the layer-skipping pattern within that regime under the same execution-native throughput feedback.

Stage I: ratio search. SPIRIT instantiates Stage I as a confidence-guided zooming search over the ratio axis, using lightweight layer-skipping probes to evaluate ratios online. Rather than treating ratio selection as flat enumeration, the controller exploits the ratio axis as a structured one-dimensional control space and progressively localizes a high-performing regime for fine-grained refinement.

Using the throughput-aligned controller score Q_w, SPIRIT maintains a ratio set $\mathcal{R}_w$ under active comparison, together with a sample-mean score estimate $\bar{\Phi}_w(r)$ and confidence bounds $U_w(r), L_w(r)$ for each ratio. Let $r_w^\dagger \in \mathcal{R}_w$ denote the incumbent ratio, i.e., the ratio with the largest current estimate. The key object is the *incumbent-centered unresolved bracket*, defined by the nearest left and right ratios whose upper confidence bound still exceeds the incumbent's lower confidence bound:

$$\begin{aligned} \ell_w &= \max\{\, r \in \mathcal{R}_w : r < r_w^\dagger,\ U_w(r) \geq L_w(r_w^\dagger)\,\}, \\ u_w &= \min\{\, r \in \mathcal{R}_w : r > r_w^\dagger,\ U_w(r) \geq L_w(r_w^\dagger)\,\}, \end{aligned} \tag{10}$$

with boundary values used when one side has no unresolved challenger. This bracket isolates the locally unresolved ratio region around the incumbent at update step w. The zooming update then operates only within this unresolved bracket:

$$\mathcal{I}_w = [\ell_w, u_w], \qquad r_{w+1} = \begin{cases} (\ell_w + r_w^\dagger)/2, & r_w^\dagger - \ell_w \geq u_w - r_w^\dagger, \\ (r_w^\dagger + u_w)/2, & \text{otherwise.} \end{cases} \tag{11}$$

Thus, the next challenger is always inserted on the wider unresolved side of the incumbent-centered bracket. Stage I therefore realizes a confidence-guided measure–compare–zoom procedure: coarse probes first nominate an incumbent, one-sided confidence comparisons progressively rule out clearly inferior regions, and midpoint challengers are introduced only within the remaining unresolved bracket. Rather than stopping at a transient lead, SPIRIT switches to conservative edge probing when the bracket becomes highly asymmetric or touches the search boundary, and commits only after the bracket has sufficiently contracted and the incumbent remains stably ordered against its nearest challengers. We provide implementation details for this search stage in Appendix B.2.

Stage II: fixed-ratio layer refinement. Once Stage I locks a stable ratio $r^\star$, SPIRIT restricts subsequent adaptation to the fixed-ratio family $c_{w+1} = \pi(z_{w+1}; r^\star)$, where only the skip-layer pattern continues to evolve. Fixing $r^\star$ largely removes the first-order consistency–speed trade-off, reducing the remaining search to pattern-level throughput optimization. Compared with the heterogeneous cross-ratio search of Stage I, this within-regime landscape is more amenable to model-based refinement, motivating a hybrid proposal policy.

$$\pi_L(z_{w+1}; r^\star) \sim \begin{cases} \pi_{\mathrm{BO}}(\cdot \mid \mathcal{D}_w, r^\star), & \text{with probability } p_{\mathrm{BO}}, \\ \pi_{\mathrm{rand}}(\cdot \mid r^\star), & \text{with probability } 1 - p_{\mathrm{BO}}. \end{cases} \tag{12}$$

where $\mathcal{D}_w = \{(L_j, Q_j)\}_{j \leq w,\ r_j = r^\star}$ denotes the accumulated fixed-ratio observations. Random proposals provide the main source of local coverage and hedge against surrogate-induced over-commitment to suboptimal patterns, while occasional Bayesian proposals exploit $\mathcal{D}_w$ to inject throughput-guided candidates. At each runtime-window boundary, SPIRIT records the score of the current pattern, updates the fixed-ratio search state, and deploys the next proposal in the following window. Refinement terminates once the incumbent pattern stabilizes or the within-request budget is exhausted, after which the best observed pattern is retained for the remainder of the request.

3.5 Cross-Request Shift-Aware Profile Adaptation

Within-request adaptation improves the active configuration for the current request, but serving workloads are often non-stationary: regimes recur, drift, and reappear over time. Re-optimizing from

Table 1: **Stationary LLaMA-family results.** SPIRIT achieves the highest Speedup across model scales and tasks. Speedup denotes Overall Speedup over AR.

Model	Method	CNN/DM			GSM8K-5shot			TinyStories		
		M	α	Speedup	M	α	Speedup	M	α	Speedup
LLaMA-2-13B-Chat	SWIFT	3.11	.77	1.19×	3.39	.87	1.21×	2.65	.80	1.10×
	D&V	3.74	.90	1.33×	3.04	.94	1.20×	3.11	.86	1.16×
	SPIRIT	3.04	.90	**1.37**×	3.10	.91	**1.27**×	3.12	.81	**1.20**×
LLaMA-2-13B	SWIFT	4.40	.95	1.46×	3.60	.97	1.40×	3.31	.97	1.36×
	D&V	5.17	.97	1.54×	3.50	.98	1.39×	5.17	.99	1.46×
	SPIRIT	6.33	.96	**1.67**×	3.97	.97	**1.49**×	9.05	.97	**1.68**×
LLaMA-2-70B	SWIFT	2.93	.96	1.29×	4.45	.98	1.39×	6.31	.96	1.45×
	D&V	4.05	.98	1.41×	4.01	.99	1.42×	4.01	.99	1.49×
	SPIRIT	6.47	.98	**1.50**×	5.39	.98	**1.57**×	9.32	.99	**1.74**×

Table 2: **Code generation on HumanEval.** Pass@1 uses greedy decoding; pass@10 uses sampling with temperature 0.6. Speedup denotes Overall Speedup over AR.

Setting	Method	CodeLLaMA-13B				CodeLLaMA-34B			
		M	α	Acc.	Speedup	M	α	Acc.	Speedup
pass@1	AUTOREGRESSIVE	1.00	–	.341	1.00×	1.00	–	.481	1.00×
	SWIFT	4.14	.94	.341	1.24×	3.24	.95	.481	1.24×
	SPIRIT	5.07	.98	.341	**1.40**×	3.86	.94	.481	**1.47**×
pass@10	AUTOREGRESSIVE	1.00	–	.701	1.00×	1.00	–	.743	1.00×
	SWIFT	2.83	.93	.621	1.15×	2.98	.89	.774	1.18×
	SPIRIT	3.58	.94	.634	**1.28**×	2.97	.87	.756	**1.32**×

scratch repeatedly relearns previously discovered structure, whereas freezing a single configuration ignores workload shifts. SPIRIT closes this gap through *cross-request profile adaptation*, maintaining a profile bank that turns locally effective configurations into reusable serving regimes. After within-request adaptation stabilizes on a locally effective regime, SPIRIT instantiates a profile p by pairing the resulting configuration $c^{(p)} = (r^{(p)}, L^{(p)})$ with the statistics $(\mu_S^{(p)}, \sigma_S^{(p)}, \mu_\alpha^{(p)}, \sigma_\alpha^{(p)})$, summarizing the mean and standard deviation of request-head throughput and draft acceptance aggregated from prior requests served under that profile. A profile thus distills recurring workload structure into a reusable decoding signature, allowing SPIRIT to reinstate an effective regime when similar early-window behavior reappears, rather than rediscovering it online.

The request head provides the earliest and cheapest evidence of whether the active profile still explains the current workload. SPIRIT therefore compresses the opening runtime windows of a new request into a mismatch score under profile p. Let $z_{S,i}^{(p)} = (S_i^{\text{head}} - \mu_S^{(p)})/\sigma_S^{(p)}$ and $z_{\alpha,i}^{(p)} = (\alpha_i^{\text{head}} - \mu_\alpha^{(p)})/\sigma_\alpha^{(p)}$ denote the standardized deviations of request-head throughput and draft acceptance. It then defines

$$d_i(p) = \left|z_{S,i}^{(p)}\right| + \omega\left|z_{\alpha,i}^{(p)}\right|. \tag{13}$$

Crucially, $d_i(p_a)$ quantifies profile mismatch rather than absolute request quality: it measures how far the request head drifts from the behavioral signature of the active profile p_a. When the mismatch grows, SPIRIT does not restart immediately. Instead, it probes the active profile against a small set of historical candidates on the same request head, ranking each candidate p by its relative probe merit:

$$J_i(p) = \left(\bar{S}_i^{\text{probe}}(p) - \bar{S}_i^{\text{probe}}(p_a)\right) + \kappa\left(\bar{\alpha}_i^{\text{probe}}(p) - \bar{\alpha}_i^{\text{probe}}(p_a)\right). \tag{14}$$

A candidate is eligible only if it improves probe merit and reduces mismatch sufficiently under a throughput-safety gate. SPIRIT then either retains the active profile, switches to a better historical profile, briefly refines after switching, or restarts online search when no reusable profile remains predictive. Cross-request adaptation is therefore evidence-gated reuse rather than blind reuse. Details on fixed-ratio refinement and the full controller appear in Appendices B.3 and B.4.

4 Experiments

4.1 Setup

Workloads. We evaluate SPIRIT on a benchmark suite covering stationary generation and non-stationary mixed-stream serving. For stationary general generation, we evaluate LLaMA-2-13B,

Table 3: **Non-stationary mixed-stream serving.** A 600-request stream alternates TinyStories, CNN/DM, and GSM8K-5shot every 100 requests. Speedup denotes Overall Speedup over AR.

Model	Method	M	α	Speedup
LLaMA-2-13B	SWIFT	2.83	.95	1.34×
	SPIRIT	5.29	.97	**1.57**×
LLaMA-2-70B	SWIFT	4.60	.92	1.36×
	SPIRIT	6.88	.97	**1.60**×

Table 4: **Ablating SPIRIT's three core mechanisms.** A1 replaces throughput scoring with acceptance-only scoring, A2 fixes $r = 0.5$, and A3 disables profile reuse. A1/A2 apply to stationary tasks, A3 to mixed streams; Avg. Drop is the mean relative drop where applicable.

Variant	LLaMA-2-13B			LLaMA-2-70B			Mixed Stream		Avg. Drop
	CNN/DM	TinyStories	GSM8K	CNN/DM	TinyStories	GSM8K	13B	70B	
Full SPIRIT	**1.67**×	**1.68**×	**1.49**×	**1.50**×	**1.74**×	**1.57**×	**1.57**×	**1.60**×	–
A1: acceptance-only	1.32×	1.32×	1.31×	1.19×	1.28×	1.17×	–	–	−21.2%
A2: fixed $r = 0.5$	1.50×	1.66×	1.42×	1.26×	1.68×	1.43×	–	–	−7.4%
A3: no profile reuse	–	–	–	–	–	–	1.43×	1.45×	−9.1%

LLaMA-2-70B, and LLaMA-2-13B-Chat on CNN/DailyMail summarization [30], TinyStories open-ended generation [31], and GSM8K mathematical reasoning [32], using 1-shot prompting for CNN/DailyMail and TinyStories and 5-shot prompting for GSM8K [33]. For code-specialized generation, we evaluate CodeLLaMA-13B and CodeLLaMA-34B [34] on HumanEval [35] with greedy decoding for pass@1 and sampling at temperature 0.6 for pass@10. For non-stationary serving, we construct mixed request streams by alternating task segments, as described in Section 4.2.

Baselines. We compare SPIRIT against standard autoregressive decoding (AR) and two training-free self-speculative baselines. D&V represents offline profile selection via Bayesian optimization over fixed layer-skipping configurations [21]. SWIFT is the strongest available open-source online baseline with an agreement-based surrogate [22]; CLaSp [23] is omitted due to unavailable public code.

Evaluation metrics. We report decode-only performance, excluding prefill [36]. The primary metric is Overall Speedup, defined as aggregate generated-token throughput relative to autoregressive decoding. We also report M, the mean committed tokens per speculative step, and α, the draft acceptance rate; detailed experimental settings and metric definitions are provided in Appendix C.

4.2 Main Results Across Serving Regimes

Stationary LLaMA-family workloads. Across the nine model–task settings in Table 1, SPIRIT consistently achieves the highest speedup, reaching 1.68× on LLaMA-2-13B and 1.74× on LLaMA-2-70B. The gain is not merely an acceptance effect: on LLaMA-2-70B TinyStories, D&V matches SPIRIT's $\alpha = 0.99$ but reaches only 1.49× Speedup, while SPIRIT increases M from 4.01 to 9.32 and achieves 1.74×. This gap exposes the objective mismatch: agreement is only a proxy, while throughput depends on how efficiently drafted work becomes committed tokens.

Transfer to code-specialized models. Table 2 evaluates transfer to code-specialized generation using CodeLLaMA-13B and CodeLLaMA-34B on HumanEval. SPIRIT achieves the highest Overall Speedup in all four model–decoding settings. On pass@1, SPIRIT improves over SWIFT from 1.24× to 1.40× on CodeLLaMA-13B and from 1.24× to 1.47× on CodeLLaMA-34B, with unchanged accuracy. Under pass@10 sampling, SPIRIT also improves speedup while maintaining comparable accuracy. These results show that throughput-aligned control is not tied to natural-language generation, but transfers to code-specialized workloads with different token statistics and acceptance patterns.

Non-stationary mixed-stream serving. Table 3 evaluates a controlled non-stationary stream instantiated as recurring task shifts: the stream cycles over TinyStories, CNN/DM, and GSM8K-5shot, with 100 requests in each task phase. This setting stresses the cross-request loop: SPIRIT detects workload shifts, reuses or refines stored profiles when regimes recur, and restarts optimization only when no suitable profile exists. SPIRIT achieves 1.57× Overall Speedup on LLaMA-2-13B and 1.60× on LLaMA-2-70B, compared with SWIFT's 1.34× and 1.36×. Thus, SPIRIT does not merely optimize isolated stationary tasks; it exploits recurring workload structure as serving-time memory.

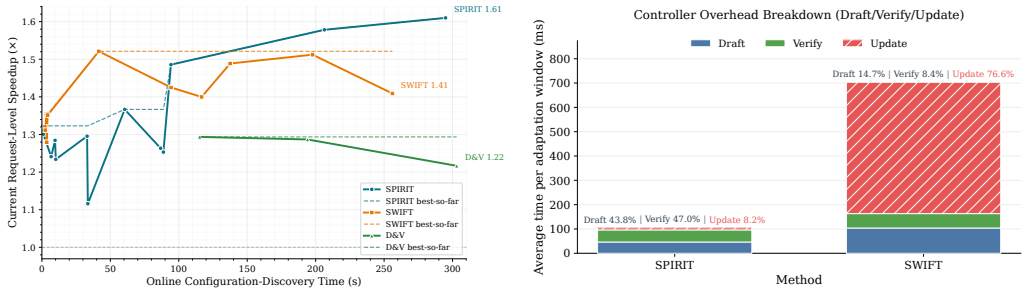


Figure 3: **Online discovery and controller cost. Left:** configurations discovered during optimization are re-evaluated on the same probe set; solid curves show current-choice speedup and dashed curves show best-so-far speedup. **Right:** average per-window optimization cost, split into draft, verify, and update; SPIRIT is step-normalized because each window contains multiple speculative steps.

4.3 Online Discovery Efficiency and Controller Cost

A serving-time controller must discover high-throughput configurations under a limited online budget while keeping adaptation cheap. We isolate this process on LLaMA-2-13B / CNN/DM-1shot: the first 13 requests are used for online optimization, and every recorded configuration is replayed on the same 20-request probe set (requests 14–33), scored by aggregate generated-token throughput relative to autoregressive decoding. This measures discovery speed and selection fidelity under true throughput. Figure 3(a) shows that SPIRIT reaches both the highest best-so-far speedup ($1.61\times$ vs. $1.52\times$ for SWIFT and $1.29\times$ for D&V) and the best final deployed configuration ($1.61\times$ vs. $1.41\times$ and $1.22\times$). SWIFT improves quickly early on, but later selects a current configuration below its own historical best. This current–best gap is not merely instability; it reveals objective misalignment. Agreement-oriented matchness can misrank configurations because neither acceptance nor skip ratio fully determines speed: even at the same ratio, different Attention/MLP patterns induce different execution graphs, KV-cache traffic, and kernel scheduling costs [37]. SPIRIT instead turns each serving window into its own measurement, ranking configurations by the committed-token throughput realized under the current prefix and execution pattern.

Figure 3(b) shows that SPIRIT is also much cheaper to adapt: update accounts for only 8.2% of adaptation-window time, versus 76.6% for SWIFT. This difference follows from the feedback interface: SPIRIT scores completed-window runtime counters without a separate surrogate pass, whereas SWIFT's update performs KV-cache cloning, an extra matchness forward, and candidate registration. Moreover, SPIRIT amortizes each update over a window of speculative steps ($N_w \approx 6$), whereas SWIFT updates nearly every step ($N_w \approx 1$), so SWIFT triggers a heavier update path more frequently and its Bayesian-optimization cost grows with accumulated observations. The breakdown also shows why draft execution must be modeled directly: draft and verify occupy comparable shares in SPIRIT (43.8% vs. 47.0%), and within SWIFT's non-update execution, draft is even more dominant (63.0% vs. 35.9%). Thus the draft path is not negligible side work: throughput depends on both target-model consistency and the intrinsic speed effects of skip ratio and concrete Attention/MLP patterns, and SPIRIT captures this trade-off while paying far lower online control cost.

4.4 Targeted Ablations

Table 4 decomposes SPIRIT into its objective, search hierarchy, and cross-request memory. Replacing the throughput-aligned objective with acceptance-only scoring (A1) causes the largest degradation, reducing average speedup by 18.4% on 13B and 24.3% on 70B; agreement alone is therefore an unreliable proxy for serving throughput. Fixing $r = 0.5$ (A2) also hurts, especially on 70B: mean speedup drops from $1.60\times$ to $1.46\times$, confirming ratio search as the first-order control stage before layer-pattern refinement. Disabling profile reuse (A3) reduces mixed-stream speedup from $1.57\times$ to $1.43\times$ on 13B and from $1.60\times$ to $1.45\times$ on 70B, showing profile reuse and task-specific refinement matter across shifts. Together, SPIRIT's gains come from aligning objective, search granularity, and temporal reuse with realized throughput, rather than from any single layer-skipping heuristic.

4.5 Generalization Across Architectures and Execution Regimes

We extend the main evaluation across architectures, workloads, batch sizes, and draft precisions, retaining full-target verification. Overall Speedup follows Appendix C.4; online timings include exploration and profile updates.

Online adaptation with NF4 self-drafts. NF4 [38] provides a 4-bit view of the target's own MLP weights without training or a separate drafter. On LLaMA-2-13B, SPIRIT optimizes (r, L) separately for FP16 and NF4 self-drafts. Table 5 shows that SPIRIT with NF4 achieves up to $2.139\times$ speedup over AR and $1.981\times$ overall, outperforming FP16 SPIRIT by 23.6% in overall throughput.

Table 5: **NF4 self-drafts improve SPIRIT throughput on LLaMA-2-13B.** Both variants retain the same FP16 verifier. Brackets show 95% CIs from request-paired bootstrap (20,000 resamples).

Task	FP16/AR	NF4/AR	NF4/FP16
CNN/DM	$1.668\times$ $[1.635, 1.699]$	$\mathbf{2.026}\times$ $[1.984, 2.063]$	$1.215\times$ $[1.197, 1.238]$
GSM8K	$1.473\times$ $[1.451, 1.494]$	$\mathbf{1.781}\times$ $[1.746, 1.813]$	$1.209\times$ $[1.186, 1.241]$
TinyStories	$1.678\times$ $[1.641, 1.697]$	$\mathbf{2.139}\times$ $[2.105, 2.162]$	$1.275\times$ $[1.253, 1.298]$
Overall	$1.603\times$ $[1.584, 1.628]$	$\mathbf{1.981}\times$ $[1.964, 2.017]$	$1.236\times$ $[1.214, 1.263]$

Modern dense GQA and sparse MoE. Table 6 extends SPIRIT to Qwen2.5 [39], Qwen3 [40], and Mixtral [41]; NF4 improves throughput over their BF16 self-drafts by 4.1–5.5%. Mixtral uses top-1 routing for drafts and its original top-2 routing for verification. Architecture and precision reshape the draft quality–cost frontier; SPIRIT adapts by directly optimizing committed tokens per unit time.

Table 6: **Online transfer to modern architectures.** Overall Speedup over AR is aggregated across CNN/DM, GSM8K, and TinyStories; full-model verification remains in BF16.

Model	BF16 self-draft	NF4 self-draft
Qwen2.5-7B	$1.304\times$ $[1.266, 1.342]$	$\mathbf{1.358}\times$ $[1.337, 1.384]$
Qwen3-8B-Base	$1.273\times$ $[1.242, 1.295]$	$\mathbf{1.343}\times$ $[1.332, 1.356]$
Mixtral-8×7B	$1.240\times$ $[1.220, 1.260]$	$\mathbf{1.296}\times$ $[1.289, 1.311]$

Broader workloads and batched serving. SPIRIT outperforms SWIFT on all six SpecBench workloads [12] and in all three controller seeds (Table 7). Their shared self-speculative backend isolates online configuration selection. SPIRIT achieves $1.462\times$ full-stream speedup versus SWIFT's $1.247\times$, improving throughput by 17.2%.

Table 7: **SpecBench on LLaMA-2-13B.** Post-adaptation speedup over AR (three-seed mean±SD).

Method	Translation	Summ.	QA	Math	RAG	Multi-turn	Overall
SWIFT	$1.291 \pm .017$	$1.394 \pm .018$	$1.296 \pm .072$	$1.219 \pm .066$	$1.292 \pm .021$	$1.278 \pm .013$	$1.276\times$
SPIRIT	$\mathbf{1.444} \pm .037$	$\mathbf{1.559} \pm .013$	$\mathbf{1.475} \pm .011$	$\mathbf{1.562} \pm .006$	$\mathbf{1.458} \pm .058$	$\mathbf{1.480} \pm .052$	$\mathbf{1.498}\times$

On LLaMA-2-13B, SPIRIT achieves $1.44\times$, $1.42\times$, and $1.23\times$ over batched AR at $B = 1, 2, 4$, with full-model verification and per-sequence ragged commit. As batching improves AR utilization and reshapes speculative costs, SPIRIT adapts to sustain gains beyond single-request decoding.

5 Conclusion

SPIRIT reframes training-free self-speculative decoding as closed-loop serving-time control rather than static layer-skipping selection, optimizing how efficiently speculative work is converted into committed-token throughput instead of agreement alone. By measuring this objective from productive runtime windows and adapting through hierarchical search and profile reuse, SPIRIT tracks workload-dependent throughput frontiers without retraining, auxiliary draft models, or offline profiling.

Acknowledgements and Disclosures

Funding. No third-party funding or support was received for this work.

Competing interests. The authors declare no competing interests.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

[2] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, volume 5, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html.

[3] Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for Medusa decoding. In *Proceedings of the Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022. URL https://arxiv.org/abs/2205.14135.

[5] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR, 2023. URL https://proceedings.mlr.press/v202/sheng23a.html.

[6] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, pages 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

[7] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

[8] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using Lookahead Decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 14060–14079. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

[9] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

[10] 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. URL https://arxiv.org/abs/2302.01318.

[11] Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.257. URL https://aclanthology.org/2023.findings-emnlp.257/.

[12] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7655–7671. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.findings-acl.456/.

[13] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1582–1595, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.88. URL https://aclanthology.org/2024.naacl-long.88/.

[14] 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*, pages 932–949. Association for Computing Machinery, 2024. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

[15] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7421–7432, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

[16] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

[17] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28935–28948. PMLR, 2024. URL https://proceedings.mlr.press/v235/li24bt.html.

[18] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A. Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 12622–12642. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.681/.

[19] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating LLMs via double early exiting. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/16336d94a5ffca8de019087ab7fe403f-Abstract-Conference.html.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL https://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.

[21] 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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 11263–11282. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.607/.

[22] Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. SWIFT: On-the-fly self-speculative decoding for LLM inference acceleration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=EKJhH5D5wA.

[23] Longze Chen, Renke Shan, Huiming Wang, Lu Wang, Ziqiang Liu, Run Luo, Jiawei Wang, Hamid Alinejad-Rokny, and Min Yang. CLaSp: In-context layer skip for self-speculative decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 31608–31618. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1525/.

[24] Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. URL https://arxiv.org/abs/2307.09288.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://papers.nips.cc/paper/7181-attention-is-all-you-need.

[26] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://papers.neurips.cc/paper/8212-blockwise-parallel-decoding-for-deep-autoregressive-models.

[27] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

[28] Yilin Yang, Longyue Wang, Shuming Shi, Prasad Tadepalli, Stefan Lee, and Zhaopeng Tu. On the sub-layer functionalities of transformer decoder. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4799–4811, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.432. URL https://aclanthology.org/2020.findings-emnlp.432/.

[29] Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 30–45, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.3. URL https://aclanthology.org/2022.emnlp-main.3/.

[30] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028/.

[31] Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023. URL https://arxiv.org/abs/2305.07759.

[32] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

[33] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020. URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[34] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. URL https://arxiv.org/abs/2308.12950.

[35] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

[36] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, Santa Clara, CA, 2024. USENIX Association. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

[37] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 46:1–46:15. IEEE, 2022. doi: 10.1109/SC41404.2022.00051. URL https://doi.org/10.1109/SC41404.2022.00051.

[38] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023. doi: 10.52202/075280-0441. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf.

[39] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

[40] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin

Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL `https://arxiv.org/abs/2505.09388`.

[41] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024. URL `https://arxiv.org/abs/2401.04088`.

A Limitations and Future Directions

This work focuses on training-free online adaptation for self-speculative decoding, where draft paths are induced from the target model itself and verified by the full target model. Within this setting, SPIRIT studies the runtime configuration-selection problem for layer-skipping draft paths, using skip ratios and concrete Attention/MLP skipping patterns as the main controllable variables. This scope is intentional: it keeps the framework plug-and-play, avoids auxiliary draft models, additional training, and offline task-specific profiling, and allows the controller to optimize directly from runtime feedback. The NF4 results further demonstrate that the same control principle applies to low-precision self-drafts while retaining full-target verification. Orthogonal efficiency techniques, such as KV-cache compression, memory offloading, and kernel-level optimization, are complementary to SPIRIT and can be combined with the proposed throughput-aligned controller in future work.

Our empirical study spans representative LLaMA-family and CodeLLaMA models as well as dense GQA and sparse MoE models from the Qwen and Mixtral families. The evaluation covers code generation, summarization, reasoning, and open-ended text generation, and includes all six SpecBench workloads and batched serving. The proposed throughput-aligned objective and online adaptation strategy do not rely on task-specific quality metrics or dataset-specific supervision. Broader model families, inference backends, and serving traces provide natural directions for future evaluation.

Finally, SPIRIT is designed as an inference-efficiency method for accelerating decoding under full-model verification. Since verification preserves the target model's output distribution, SPIRIT accelerates generation while leaving model behavior governed by the underlying target model and its deployment pipeline.

B SPIRIT Online Adaptation: Implementation and Algorithms

This appendix specifies the online adaptation algorithm that instantiates SPIRIT. Complementing the main paper's runtime-throughput-driven framework, we detail the mechanisms required for reproducibility: phase-calibrated window feedback for comparable online measurements, confidence-guided skip-ratio localization followed by fixed-ratio layer-pattern refinement, and shift-aware profile reuse across non-stationary request streams.

B.1 Controller Signal: Phase-Calibrated Runtime Feedback

SPIRIT drives its online controller with generated-token throughput, but raw window measurements are phase-dependent. Even under an unchanged skip configuration, per-token latency increases as the KV cache grows, making later windows appear worse for reasons unrelated to the skip ratio or layer pattern. We therefore apply a lightweight phase calibration before passing a window score to the controller. The calibrator fits a shared affine trend from effective prefix length to latency per generated token, and uses it only to re-base each window score to a common reference prefix length. Thus, calibration introduces no auxiliary reward or configuration-specific preference: skip ratios and layer patterns are still ranked by measured runtime throughput after phase re-basing.

The calibrator ingests observations $z = (m, \bar{x}, y)$, where y is the latency per generated token over the measured interval, $\bar{x}$ is the average effective prefix length, and m is a step-coverage weight. We set $m = 1$ for step-level telemetry and $m = m_w$ for a runtime window containing m_w speculative steps. The weight m is used only in the online length fit; it is not a generated-token count and does not enter the throughput objective. This convention makes step-level and window-level telemetry compatible by weighting the length fit by speculative-step coverage rather than by acceptance-dependent generated-token yield. For a speculative step that starts from prefix length p_t and commits g_t output tokens, we assign the effective prefix length

$$x_{\text{step}} = p_t + \frac{1}{2} g_t.$$

This midpoint convention assigns each step a representative prefix length for the interval over which the prefix grows from p_t to $p_t + g_t$, regardless of whether the committed tokens are accepted draft tokens or verifier bonus tokens. The telemetry granularity depends on the timing backend: with stable single-device timing, SPIRIT may ingest step-level observations $(1, x_{\text{step}}, y_t)$ for more frequent length-fit updates; with CUDA-event window timing or synchronized multi-device wall-clock timing, it emits one aggregate observation $(m_w, \bar{x}_w, y_w)$ per runtime window to avoid over-interpreting

Algorithm 1 Phase-Calibrated Runtime Target

Require: Adaptation-window observation $z_w = (S_w, x_w, y_w, m_w)$, where S_w is raw throughput, x_w is the effective prefix length, y_w is latency per generated token, and m_w is a step-coverage weight; length state $\mathcal{F} = (n, s_x, s_y, s_{xx}, s_{xy})$; previous target Q_{w-1}

Ensure: Updated length state $\mathcal{F}$, calibration factor f_{len}, and controller target Q_w

// Maintain a step-weighted affine model of phase-dependent latency.

$n \leftarrow n + m_w, \quad s_x \leftarrow s_x + m_w x_w, \quad s_y \leftarrow s_y + m_w y_w$

$s_{xx} \leftarrow s_{xx} + m_w x_w^2, \quad s_{xy} \leftarrow s_{xy} + m_w x_w y_w$

// Calibrate the current window to a reference-prefix phase.

$f_{\text{base}} \leftarrow \sqrt{(x_w + 1)/(x_{\text{ref}} + 1)}$

$f_{\text{len}} \leftarrow f_{\text{base}}$

$\Delta \leftarrow n s_{xx} - s_x^2$

if $n \geq n_{\text{warm}}$ and $\Delta > \epsilon$ **then**

  $b \leftarrow (n s_{xy} - s_x s_y)/\Delta, \quad a \leftarrow (s_y - b s_x)/n$

  $p_w \leftarrow a + b x_w, \quad p_{\text{ref}} \leftarrow a + b x_{\text{ref}}$

  if $p_w > \epsilon$ and $p_{\text{ref}} > \epsilon$ **then**

    $f_{\text{fit}} \leftarrow p_w / p_{\text{ref}}$

    $\eta \leftarrow 1$ if $n_{\text{ramp}} \leq 0$; otherwise $\eta \leftarrow \text{clip}((n - n_{\text{warm}})/n_{\text{ramp}}, 0, 1)$

    $f_{\text{len}} \leftarrow (1 - \eta) f_{\text{base}} + \eta f_{\text{fit}}$

  end if

end if

$f_{\text{len}} \leftarrow \text{clip}(f_{\text{len}}, f_{\min}, f_{\max})$

$\widetilde{S}_w \leftarrow S_w f_{\text{len}}$

$Q_w \leftarrow \widetilde{S}_w$ if Q_{w-1} is uninitialized; otherwise $Q_w \leftarrow \alpha_{\text{ema}} \widetilde{S}_w + (1 - \alpha_{\text{ema}}) Q_{w-1}$

return $\mathcal{F}, f_{\text{len}}, Q_w$

asynchronous step-level noise. In both cases, the search policy receives an updated optimization target only after a complete runtime window, so calibration may be fine-grained while controller decisions remain window-paced.

Algorithm 1 summarizes this update. The resulting Q_w is used only as the optimizer-facing target for within-request control, including ratio localization and fixed-ratio layer-pattern refinement; length calibration improves comparability across decoding phases, and EMA smoothing reduces window-level timing noise. Cross-request adaptation instead relies on request-head raw throughput and draft acceptance, without length calibration or EMA smoothing, so that shift detection remains sensitive to the active profile's immediate behavior on a new request.

B.2 Confidence-Guided Zooming Ratio Search

The main text motivates SPIRIT's ratio-first hierarchy: first localize the skip-ratio regime, then refine the layer pattern. We instantiate the outer stage as confidence-guided zooming over r, treating the skip ratio as a low-dimensional control variable and progressively concentrating productive decoding windows on ratio regions that remain competitive under the phase-calibrated window target Q_w from Appendix B.1.

SPIRIT maintains an active set of ratio ticks $\mathcal{R} \subseteq [r_{\min}, r_{\max}]$, initialized from a coarse ratio grid and refined online by inserting midpoint candidates. For each active ratio $r \in \mathcal{R}$, it keeps

$$H_r = \big(n_r, \mu_r, M_{2,r}, \bar{\alpha}_r, \mathcal{B}_r\big),$$

where n_r is the number of evaluated windows, μ_r is the online mean of the controller target Q_t, and $M_{2,r}$ accumulates the second central moment for variance estimation. The term $\bar{\alpha}_r$ is an EMA of draft acceptance, while $\mathcal{B}_r$ stores high-scoring layer patterns observed under ratio r to warm-start the subsequent fixed-ratio optimizer, rather than to select the final layer pattern during ratio search. From $(n_r, M_{2,r})$, SPIRIT computes the empirical standard error se_r and forms confidence bounds

$$\text{LCB}_r = \mu_r - c\,\text{se}_r, \qquad \text{UCB}_r = \mu_r + c\,\text{se}_r,$$

where c is a fixed confidence multiplier. The incumbent and the remaining statistically competitive challengers are defined as

$$r_{\text{inc}} = \arg\max_{r \in \mathcal{R}} \mu_r, \qquad \mathcal{C} = \{r \in \mathcal{R} \setminus \{r_{\text{inc}}\} : \text{UCB}_r \geq \text{LCB}_{r_{\text{inc}}}\}.$$

Algorithm 2 Confidence-Guided Zooming Ratio Search

Require: Active ratio ticks $\mathcal{R}$, states $\{H_r\}_{r\in\mathcal{R}}$, observation (r, L, Q_w, α_w)
Require: Minimum evidence $n_{\min}$, confirmation evidence n_{conf}, interval tolerance Δ_r, confidence multiplier c
Ensure: CONTINUE(r_{next}) or LOCK($r^\star, \mathcal{B}_{r^\star}, W_{\text{final}}, \tau_{\text{stop}}$)
Helper: MOSTUNCERTAIN($\mathcal{S}$) $= \arg\max_{r\in\mathcal{S}}(\text{UCB}_r - \text{LCB}_r)$.
Helper: INSERTFEASIBLEMIDPOINT($[a, b], \mathcal{R}$) inserts and initializes the nearest feasible midpoint in the largest unresolved active-tick gap inside $[a, b]$.
Update $H_r = (n_r, \mu_r, M_{2,r}, \bar{\alpha}_r, \mathcal{B}_r)$ with (L, Q_w, α_w)
// Obtain coarse evidence before eliminating ratio regions.
if there exists $r' \in \mathcal{R}$ with $n_{r'} < n_{\min}$ **then**
    return CONTINUE(LEASTSAMPLED($\mathcal{R}$))
end if
Compute $\text{LCB}_{r'} = \mu_{r'} - c\,\text{se}_{r'}$ and $\text{UCB}_{r'} = \mu_{r'} + c\,\text{se}_{r'}$ for all $r' \in \mathcal{R}$
$r_{\text{inc}} \leftarrow \arg\max_{r'\in\mathcal{R}} \mu_{r'}$
$\mathcal{C} \leftarrow \{r' \in \mathcal{R} \setminus \{r_{\text{inc}}\} : \text{UCB}_{r'} \geq \text{LCB}_{r_{\text{inc}}}\}$
// Lock only after conservative confirmation.
if $\mathcal{C} = \emptyset$ **then**
    if $n_{r_{\text{inc}}} \geq n_{\text{conf}}$ and EDGEVALIDATED($r_{\text{inc}}, \mathcal{R}$) **then**
        $r^\star \leftarrow r_{\text{inc}}$
        Set W_{final} from recent throughput dispersion and τ_{stop} from $\bar{\alpha}_{r^\star}$
        return LOCK($r^\star, \mathcal{B}_{r^\star}, W_{\text{final}}, \tau_{\text{stop}}$)
    end if
    return CONTINUE(EDGEORSELFPROBE($r_{\text{inc}}, \mathcal{R}$))
end if
// Otherwise zoom into the unresolved incumbent–challenger region.
$r_{\text{chal}} \leftarrow \arg\max_{r'\in\mathcal{C}} \text{UCB}_{r'}$
Let $[a, b]$ be the interval adjacent to r_{inc} selected for bisection in Eq. 11
if LARGESTGAP($[a, b], \mathcal{R}$) $> \Delta_r$ **then**
    $r_{\text{mid}} \leftarrow$ INSERTFEASIBLEMIDPOINT($[a, b], \mathcal{R}$)
    return CONTINUE(r_{mid})
end if
if $\min(n_{r_{\text{inc}}}, n_{r_{\text{chal}}}) < n_{\text{conf}}$ **then**
    return CONTINUE(LEASTSAMPLED($\{r_{\text{inc}}, r_{\text{chal}}\}$))
end if
return CONTINUE(MOSTUNCERTAIN($\{r_{\text{inc}}, r_{\text{chal}}\}$))

If challengers remain, SPIRIT zooms into the unresolved incumbent–challenger interval by either inserting a midpoint ratio or allocating the next productive window to the incumbent or challenger with less evidence or larger uncertainty. Algorithm 2 extends Eq. 11 by bisecting the largest unresolved active-tick gap within the selected side; remeasurement uses the highest-UCB challenger. If no challenger remains, SPIRIT locks the incumbent only after confirmation and edge validation, preventing low-sample noise from prematurely excluding boundary regimes.

The edge-validation routines in Algorithm 2 serve only as a final stopping guard. EDGEVALIDATED returns true for any non-endpoint incumbent; for an endpoint incumbent, the corresponding boundary-side comparison, when present, must have at least n_{conf} observations. If this requirement fails, EDGEORSELFPROBE schedules the under-measured boundary comparison; if no such comparison remains, it schedules one additional confirmation window for the incumbent. After locking $r^\star$, SPIRIT warm-starts the fixed-ratio optimizer from $\mathcal{B}_{r^\star}$, sets W_{final} from recent throughput dispersion, and derives τ_{stop} from the acceptance EMA, thereby passing both the skip-ratio decision and ratio-conditioned evidence to the residual layer-pattern search.

B.3 Fixed-Ratio Layer-Pattern Refinement

After $r^\star$ is locked, SPIRIT refines only the residual layer-placement choice within

$$\mathcal{L}(r^\star) = \{L : \rho(L) = r^\star\}.$$

Algorithm 3 Fixed-Ratio Layer-Pattern Refinement

Require: Locked ratio $r^\star$, ratio-conditioned bank $\mathcal{B}_{r^\star}$, refinement budget B
Ensure: Refined serving pattern $L^\star$ and optimizer state $\mathcal{M}_{r^\star}$

Initialize $\mathcal{M}_{r^\star}$ from trusted entries in $\mathcal{B}_{r^\star}$; set $L^\star$ to the current pattern and $Q^\star \leftarrow -\infty$
for $b = 1, \ldots, B$ **do**
  Let $L_b \in \mathcal{L}(r^\star)$ be the current serving pattern
  Evaluate L_b for one runtime feedback window and compute Q_b using Algorithm 1
  Update $\mathcal{M}_{r^\star}$ with (L_b, Q_b)
  if $Q_b > Q^\star$ **then**
    $L^\star \leftarrow L_b, \quad Q^\star \leftarrow Q_b$
  end if
  if REFINEMENTCONVERGED($\mathcal{M}_{r^\star}$) **then**
    break
  end if
  $L_{b+1} \leftarrow$ PROPOSALPOLICY($\mathcal{M}_{r^\star}, \mathcal{L}(r^\star), b$)
  Switch serving pattern to L_{b+1}
end for
Switch serving pattern to $L^\star$
return $L^\star, \mathcal{M}_{r^\star}$

Each refinement round evaluates the current pattern for one runtime feedback window, updates the optimizer with the phase-calibrated target, and deploys the next ratio-preserving proposal. PROPOSALPOLICY abstracts the proposal schedule: it interleaves coverage-oriented ratio-preserving proposals with optimizer-guided proposals, while always enforcing $L \in \mathcal{L}(r^\star)$. Here, $\mathcal{M}_{r^\star}$ denotes the fixed-ratio Bayesian optimizer state, including the evaluated patterns, their phase-calibrated targets, and the surrogate used for BO proposals. REFINEMENTCONVERGED abstracts the refinement budget and no-improvement stopping criteria.

B.4 End-to-End Online Controller: Optimize, Monitor, and Reuse

The previous subsections describe how SPIRIT builds a serving configuration from runtime feedback by localizing the skip-ratio regime and refining the layer pattern within the locked ratio. Across a non-stationary request stream, SPIRIT wraps this builder in an optimize–monitor–reuse lifecycle: it builds a profile when no reliable one is active, serves subsequent requests with the active profile while monitoring only the request head, and reopens adaptation only when the head behavior departs from the profile's stored statistics. A serving profile is a reusable controller state:

$$\pi = \big(r^\star, L^\star, W_{\text{final}}, \tau_{\text{stop}}, \mathcal{M}_{r^\star}, \mathcal{H}_{\text{head}}\big),$$

where $r^\star$ and $L^\star$ define the draft path, W_{final} and τ_{stop} are runtime controls, $\mathcal{M}_{r^\star}$ stores the Bayesian optimizer state, and $\mathcal{H}_{\text{head}}$ stores running request-head statistics. For a request-head observation $h_i = (s_i, \alpha_i)$, with raw generated-token throughput s_i and draft acceptance α_i, SPIRIT computes

$$d_i(\pi) = \left| \frac{s_i - \mu_s}{\max(\sigma_s, \sigma_{s,\min})} \right| + \lambda_\alpha \left| \frac{\alpha_i - \mu_\alpha}{\max(\sigma_\alpha, \sigma_{\alpha,\min})} \right|,$$

where (μ_s, σ_s) and $(\mu_\alpha, \sigma_\alpha)$ are the running mean and standard deviation of request-head throughput and draft acceptance, respectively, maintained in $\mathcal{H}_{\text{head}}$. A head is compatible when $d_i(\pi)$ remains below a low-deviation threshold. A shift is triggered only after sufficient head evidence, either by one high-deviation head or by repeated weak deviations.

When a request-head shift is triggered, SPIRIT performs bounded, non-committing evaluations of candidate historical profiles; the measurements are used only for adaptation-response selection. It then follows a least-cost ladder: STAY if the probe evidence is inconclusive, which spends no search budget; SWITCH when a stored profile is clearly better and acceptance-feasible, which reuses the profile; SWITCHANDREFINE when a compatible profile identifies the right ratio regime but still needs residual layer-pattern adjustment, which spends only a small fixed-ratio refinement budget; and RESTART only when the active profile remains strongly mismatched and no stored profile is feasible, which re-enters the full ratio-first builder. Algorithm 4 summarizes this optimize–monitor–reuse lifecycle, in which SPIRIT spends no search budget during stable serving and reopens adaptation only when profile reuse is insufficient.

Algorithm 4 End-to-End SPIRIT Online Controller

Require: Request stream $\{x_i\}$, profile bank $\mathcal{P}$, active profile π_{act} if available
Ensure: Decoding under an adaptively selected serving profile

Subroutine BUILDPROFILE(x): advance request x with productive adaptation windows. Each completed window emits Q_w using Algorithm 1; before ratio lock it updates Algorithm 2, and after ratio lock it updates Algorithm 3. Return π_{new} if a reusable profile is materialized.
Subroutine SELECTPROFILEACTION$(\pi_{\mathrm{act}}, \mathcal{P})$: run bounded, non-committing request-head trials for candidate profiles and return the least-cost supported action and profile.
Subroutine APPLYPROFILEACTION$(a, \pi_{\mathrm{sel}}, \pi_{\mathrm{act}}; x)$: apply the selected action, complete the remaining decoding of x, and return the updated active profile and any newly materialized profile.

for each request x_i **do**
  $\pi_{\mathrm{new}} \leftarrow \varnothing$
  if π_{act} is unavailable **then**
    $\pi_{\mathrm{new}} \leftarrow$ BUILDPROFILE(x_i)
    if π_{new} is ready **then**
      $\pi_{\mathrm{act}} \leftarrow \pi_{\mathrm{new}}$; insert π_{new} into $\mathcal{P}$
    end if
    continue
  end if
  Decode a bounded request head using π_{act}
  Measure $h_i^{\mathrm{act}} = (s_i^{\mathrm{act}}, \alpha_i^{\mathrm{act}})$ and compute $d_i(\pi_{\mathrm{act}})$
  if COMPATIBLE$(d_i(\pi_{\mathrm{act}}))$ **then**
    Update $\mathcal{H}_{\mathrm{head}}(\pi_{\mathrm{act}})$ with h_i^{act}
    Continue decoding the remaining tokens using π_{act}
    continue
  end if
  $(a, \pi_{\mathrm{sel}}) \leftarrow$ SELECTPROFILEACTION$(\pi_{\mathrm{act}}, \mathcal{P})$
  $(\pi_{\mathrm{act}}, \pi_{\mathrm{new}}) \leftarrow$ APPLYPROFILEACTION$(a, \pi_{\mathrm{sel}}, \pi_{\mathrm{act}}; x_i)$
  if π_{new} is ready **then**
    $\pi_{\mathrm{act}} \leftarrow \pi_{\mathrm{new}}$; insert π_{new} into $\mathcal{P}$
  end if
end for

B.5 Implementation Protocol

We instantiate Algorithms 1–4 with the protocol constants in Table 8, which specify the search range, phase-calibration rule, ratio-locking criterion, fixed-ratio refinement budget, and request-head shift detector. We include these constants to make the controller logic reproducible while omitting backend-specific engineering choices, such as CUDA event placement and device-synchronization details, that affect measurement implementation but not the controller rules themselves. These constants are fixed before evaluation and do not introduce task-specific rewards: within-request candidates are ranked by the phase-calibrated generated-token throughput Q_w, while cross-request shift handling uses raw request-head throughput and draft acceptance only for profile selection.

To preserve draft quality, we keep the first and last transformer layers active by default; N denotes the number of remaining layers eligible for skipping, reindexed as $\ell = 1, \ldots, N$. We use the same skip-ratio range across all workloads, treating it as a fixed feasible search range rather than a task-specific choice. The request-head deviation score uses raw throughput and draft acceptance only to trigger profile reuse, refinement, or rebuilding; it does not modify the within-request target Q_w used for ratio and layer search. For fixed-ratio refinement, B is an upper budget with shared no-improvement early stopping; it bounds online adaptation cost and does not change the throughput objective used to compare candidate layer patterns.

Together, these protocol choices make SPIRIT a window-paced online controller: phase calibration makes runtime feedback comparable across decoding phases, confidence-bounded zooming localizes the skip-ratio regime, fixed-ratio refinement searches residual layer placements, and request-head monitoring amortizes adaptation through profile reuse.

Table 8: Core protocol constants used to instantiate the SPIRIT online adaptation algorithms.

Algorithm component	Quantity	Value / rule
Search space	Ratio range $[r_{\min}, r_{\max}]$	$[0.40, 0.50]$
	Skip budget	$\lvert L\rvert = \lfloor 2Nr \rfloor$
Phase-calibrated target	Reference prefix length x_{ref}	$\max(32, p_0)$, post-prefill prefix length
	Warmup / ramp	$n_{\mathrm{warm}} = 8,\ n_{\mathrm{ramp}} = 8$
	Length-factor clipping	$f_{\min} = 0.5,\ f_{\max} = 2.0$
	Target EMA	$\alpha_{\mathrm{ema}} = 0.2$
Zooming ratio search	Narrowing tolerance	$\Delta_r = 0.03$
	Minimum evidence	$n_{\min} = 3$ runtime windows per active ratio
	Confirmation evidence	$n_{\mathrm{conf}} = 5$ runtime windows
	Confidence multiplier	$c = 2.0$
Fixed-ratio refinement	Refinement budget B	500 runtime feedback windows after ratio locking
	Proposal policy	$p_{\mathrm{BO}} = 0.04$ Bayesian proposal; otherwise random proposal
End-to-end controller	Request-head monitor	$H_{\mathrm{head}} = 2$ runtime windows
	Acceptance-deviation weight	$\lambda_\alpha = 0.7$
	Deviation floors	$\sigma_{s,\min} = 10^{-3},\ \sigma_{\alpha,\min} = 10^{-3}$
	Strong shift trigger	$d_i(\pi) > 2.6$
	Weak shift trigger	$d_i(\pi) > 1.1$ for 3 consecutive heads

Table 9: Hardware allocation by model scale.

Model scale	GPU allocation	Total GPU memory	Execution mode
13B	1× NVIDIA RTX A6000	48GB	Single GPU
34B	1× NVIDIA A100 PCIe	80GB	Single GPU
70B	4× NVIDIA RTX A6000	192GB	Model parallel

C Experimental Details and Reproducibility

C.1 Evaluation Control and Runtime Environment

All comparisons are controlled within each model–task setting: compared methods use the same target checkpoint, hardware allocation, numerical precision, prompts, generation parameters, request ordering, stochastic-decoding randomness when applicable, and timing protocol. Dataset shuffling, demonstration selection, and request ordering are fixed across methods so that performance comparisons are paired under identical inputs. Unless otherwise specified in Section 4.5, all experiments use batch size one, matching the standard single-request setting in speculative decoding evaluations.

We run experiments on a heterogeneous two-server cluster equipped with Intel Xeon Gold 6430 CPUs. Table 9 summarizes the hardware allocation by model scale. For each scale, all compared methods are evaluated under the same allocation: 13B models use one NVIDIA RTX A6000 GPU, 34B models use one NVIDIA A100 PCIe GPU, and the 70B model uses model-parallel execution across four NVIDIA RTX A6000 GPUs.

The software environment uses Ubuntu 22.04.5 LTS, NVIDIA driver 580.126.20, nvcc 13.0, PyTorch 2.9.0+cu128, and HuggingFace Transformers 4.33.3. All models are evaluated in FP16 unless otherwise specified in Section 4.5. We use HuggingFace Transformers together with a custom self-speculative decoding backend. The 13B and 34B models use single-GPU execution, while the 70B model uses HuggingFace automatic device mapping for model-parallel execution.

For reported wall-clock results, we synchronize CUDA operations and use the same timing protocol for all methods. For SPIRIT's runtime-window feedback, we use CUDA events on single-GPU runs and synchronized wall-clock timing on multi-GPU 70B runs.

The NF4 self-draft adds 2.8 GB of memory on Qwen3-8B-Base and 4.4 GB (17%) on LLaMA-2-13B. One-time initialization takes approximately 27 s and 40 s, respectively, and is excluded from decode timing.

Table 10: Dataset and evaluation settings. Demonstrations, examples, and request order are fixed across compared methods.

Setting	Data / protocol	Prompting	Size
CNN/DM	`cnn_dailymail` 3.0.0, test split	1-shot from train	1000
GSM8K	`openai/gsm8k` main, test split	5-shot from train	1000
TinyStories	`roneneldan/TinyStories`, train split	1-shot from train	1000
HumanEval	OpenAI HumanEval, all problems	original prompt	164
Streaming	TinyStories, GSM8K, and CNN/DM; two cycles, 100 requests per segment	as above	600

C.2 Models, Workloads, and Generation Settings

We evaluate the following HuggingFace model checkpoints from the LLaMA-2 and CodeLLaMA model families [24; 34]: `meta-llama/Llama-2-13b-hf`, `meta-llama/Llama-2-13b-chat-hf`, `meta-llama/Llama-2-70b-hf`, `meta-llama/CodeLlama-13b-hf`, and `meta-llama/CodeLlama-34b-hf`. All model checkpoints and datasets used in our evaluation are accessed from their public release channels and used according to their respective licenses and access terms.

LLaMA-2 models are evaluated on CNN/DM [30], GSM8K [32], and TinyStories [31]. CodeLLaMA models are evaluated on HumanEval [35] under pass@1 and pass@10 settings. Table 10 summarizes the workload settings.

TinyStories is used as an open-ended continuation workload rather than as a held-out accuracy benchmark. HumanEval is loaded through `human_eval.data.read_problems()` and evaluated on all 164 problems.

Unless otherwise specified, all generations use `max_new_tokens=512`. CNN/DM, GSM8K, TinyStories, and HumanEval pass@1 use greedy decoding with temperature 0.0. HumanEval pass@10 uses sampling with temperature 0.6, `top_p=0.95`, and ten samples per task.

CNN/DM uses a 1-shot article–summary prompt with `Article:` and `Summary:` fields, GSM8K uses a 5-shot question–answer prompt with `Question:` and `Answer:` fields, and TinyStories uses a 1-shot story-continuation prompt with `Story:` and `Continue:` fields. For HumanEval, we use the original benchmark prompt field without additional instructions. Generation stops at EOS or the maximum generation length. For HumanEval, we do not apply task-specific code stop tokens such as `\nclass`, `\ndef`, or code-block delimiters. Post-processing only applies trailing whitespace removal and indentation normalization.

C.3 Baselines and Protocol Constants

The autoregressive baseline uses the same target checkpoint, target-model precision, maximum generation length, prompts, and decoding settings as the corresponding accelerated method, and is executed with HuggingFace `AutoModelForCausalLM.generate`. For self-speculative baselines, we evaluate Draft & Verify and SWIFT under the same model checkpoints, precision, generation settings, and timing protocol. Draft & Verify is evaluated as a fixed-profile self-speculative baseline, while SWIFT performs online layer-set adaptation following its agreement-oriented objective.

SPIRIT uses the same self-speculative decoding backend and augments it with runtime ratio search and online layer-pattern refinement. SPIRIT and SWIFT share the same KV-cache implementation, stopping rule, and dynamic draft-tree construction, so their comparison primarily reflects the online configuration-selection strategy.

SPIRIT aggregates runtime feedback over a fixed number W of speculative decoding steps. Unless otherwise stated, we use $W = 6$ for 13B models and $W = 3$ for 34B and 70B models, including CodeLLaMA-34B HumanEval. These values are chosen a priori by model scale and kept fixed throughout evaluation. The default draft stopping threshold is 0.8; fixed-configuration runs use 0.9 only when explicitly specified. For Bayesian layer-pattern proposals, we use a Gaussian-process surrogate with $\alpha = 10^{-2}$ and a UCB acquisition function parameterized by $\kappa = 2.5$ and $\xi = 0.0$. Other controller constants are summarized in Table 8.

C.4 Metrics

We report aggregate generated-token throughput,

$$\mathrm{TPS} = \frac{\sum_i G_i}{\sum_i T_i},$$

where G_i is the number of committed output tokens for request i and T_i is the measured wall-clock generation time. Overall speedup is computed relative to autoregressive decoding:

$$\mathrm{Speedup} = \frac{\sum_i G_i / \sum_i T_i}{\sum_i G_i^{\mathrm{AR}} / \sum_i T_i^{\mathrm{AR}}}.$$

This aggregate definition reflects serving throughput over the evaluated request stream and is not an average of per-request speedups.

For self-speculative decoding, let N_i be the number of speculative steps, $A_{d,i}$ the number of accepted draft tokens, and D_i the number of drafted tokens. We report

$$M = \frac{\sum_i G_i}{\sum_i N_i}, \qquad \alpha = \frac{\sum_i A_{d,i}}{\sum_i D_i}.$$

Here, M includes the verified bonus token committed at each speculative step, while α measures draft-token acceptance and excludes the bonus token.

C.5 Repeated End-to-End Measurements

To assess run-to-run stability, we repeat SPIRIT, SWIFT, and D&V independently three times in each of the six stationary 13B settings in Table 1. All runs follow the same FP16, batch-one evaluation protocol, with Overall Speedup computed as in Appendix C.4.

Table 11: **Repeated 13B results.** Overall Speedup over AR (mean±SD across three independent end-to-end runs per method). Standard deviations measure run-to-run variation.

Model	Task	SWIFT	D&V	SPIRIT
LLaMA-2-13B	CNN/DM	$1.451 \pm 0.007\times$	$1.535 \pm 0.004\times$	$\mathbf{1.652} \pm 0.031\times$
	GSM8K	$1.396 \pm 0.006\times$	$1.381 \pm 0.012\times$	$\mathbf{1.468} \pm 0.037\times$
	TinyStories	$1.344 \pm 0.014\times$	$1.456 \pm 0.007\times$	$\mathbf{1.663} \pm 0.026\times$
LLaMA-2-13B-Chat	CNN/DM	$1.181 \pm 0.008\times$	$1.318 \pm 0.013\times$	$\mathbf{1.343} \pm 0.043\times$
	GSM8K	$1.203 \pm 0.011\times$	$1.197 \pm 0.009\times$	$\mathbf{1.241} \pm 0.035\times$
	TinyStories	$1.092 \pm 0.013\times$	$1.152 \pm 0.011\times$	$\mathbf{1.191} \pm 0.027\times$

Table 11 shows that SPIRIT retains the highest mean Overall Speedup in all six settings, reproducing its lead over both baselines across base and chat checkpoints.

C.6 Warmup and Cumulative Throughput

We measure cumulative throughput from the first request as total generated tokens divided by total generation time, including online adaptation; speedup follows Appendix C.4. Cumulative break-even is the earliest request from which SPIRIT's cumulative throughput matches or exceeds AR's for the remainder of the evaluated stream.

Table 12: **Cumulative warmup on LLaMA-2.** Speedup over AR through request k.

Model	$k=1$	$k=5$	$k=10$	$k=20$	Cumulative break-even
LLaMA-2-13B	$0.977\times$	$1.277\times$	$1.341\times$	$\mathbf{1.507}\times$	Request 2
LLaMA-2-70B	$0.928\times$	$1.183\times$	$1.310\times$	$\mathbf{1.453}\times$	Request 3

Table 12 shows that SPIRIT reaches cumulative break-even within two to three requests despite initial adaptation costs. Under a stable workload, configuration reuse spreads the initial search cost over subsequent requests, so long-run speedup is governed by the selected configuration's sustained throughput advantage rather than the transient cost of discovering it.

C.7 Sustained Decode Acceleration

SPIRIT's execution-coupled search produces committed tokens while identifying the best-performing configurations for subsequent reuse. Figure 4 traces this process in an independent LLaMA-2-13B evaluation with NF4 self-drafts and full FP16 verification. Across eight runs with distinct controller seeds, each run starts from an empty controller state and serves the same ordered stream of 1,000 TinyStories requests on a single RTX A6000. AR is measured on the same GPU and requests; both methods use batch size one and a 512-token generation cap. Throughput is measured as committed output tokens per unit of generation time, excluding prefill while including online search and profile updates; model loading and one-time draft setup are excluded. The NF4 draft uses fused kernels, while full FP16 verification is retained.

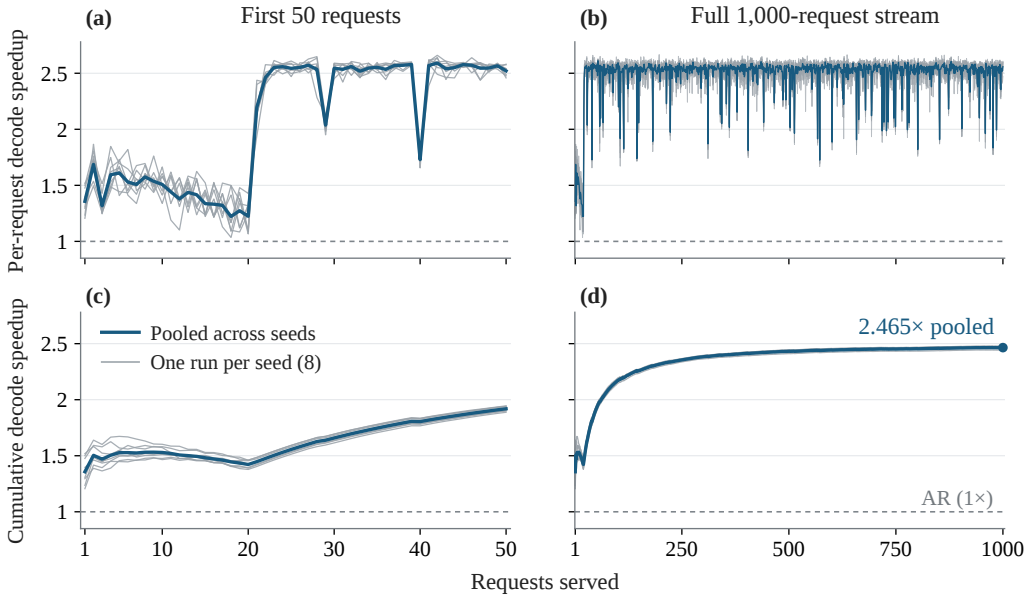


Figure 4: **Online adaptation delivers sustained throughput gains.** Per-request (a,b) and cumulative (c,d) speedup of SPIRIT over matched AR on TinyStories, using LLaMA-2-13B with NF4 self-drafts and full FP16 verification. The left column shows the first 50 requests; the right covers the complete 1,000-request stream. Gray curves show all eight individual runs; blue curves report speedup computed from each method's token and time totals pooled across runs. Timings include online search and profile updates.

SPIRIT delivers net throughput gains while its controller is still searching: cumulative throughput remains above AR throughout adaptation in all eight runs. During subsequent configuration reuse, pooled per-request decode speedup peaks at $2.612\times$. Over the complete 1,000-request stream, SPIRIT achieves a pooled speedup of $2.465\times$ across eight runs, with the best run reaching $2.478\times$. Under a stable workload, the optimized configuration's throughput advantage persists as its discovery cost becomes a diminishing fraction of total serving time.

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 present SPIRIT as training-free online adaptation for self-speculative decoding and state its throughput-oriented contributions within this scope. These claims are supported by the method and experimental results in Sections 3 and 4, with scope and reproducibility details provided in Appendices A and C.

   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: Appendix A discusses the scope and boundaries of SPIRIT as a training-free inference-efficiency method for layer-skipping self-speculative decoding, and outlines future directions including broader model families, inference backends, and serving traces.

   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 introduce new formal theoretical results or proof-based claims. The distribution-preserving property invoked in the paper follows from the standard speculative-decoding verification rule with the full target model, while SPIRIT's contributions focus on runtime control, online configuration selection, and empirical evaluation.

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: Appendix C reports the hardware allocation, software stack, model checkpoints, datasets and splits, prompting and demonstration construction, generation settings, timing protocol, baseline protocols, SPIRIT protocol constants, and metric definitions. It also specifies how dataset shuffling, demonstration selection, generation sampling, and request ordering are fixed across methods to keep comparisons controlled under identical inputs.

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: [No]

   Justification: The datasets and model checkpoints used in our experiments are accessed from public release channels. A public code release is not provided at submission time; however, Appendices B and C provide the algorithmic and experimental details needed to reproduce the main results.

   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 evaluated workloads, baselines, metrics, and overall evaluation protocol, while Appendix C details the dataset splits, prompts, generation settings, runtime-window sizes, SPIRIT controller constants, and timing protocol. SPIRIT is training-free, so no model-training hyperparameters are introduced.

   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: Appendix C.5 reports mean±standard deviation across three independent end-to-end runs for SPIRIT, SWIFT, and D&V in all six stationary 13B base/chat settings; the standard deviations measure run-to-run variation. Section 4.5 also reports three-seed SpecBench variation and request-paired bootstrap 95% confidence intervals for the LLaMA-2-13B FP16/NF4 comparison (20,000 resamples). These summaries quantify variability without asserting statistical significance for every individual comparison.

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: Appendix C.1 reports the CPU/GPU hardware, GPU memory, execution mode, software stack, precision, and timing protocol used for each model scale. The experiments are inference-only, and the reported throughput and wall-clock measurements are collected under the same synchronized timing protocol for all compared methods.

   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 studies inference-efficiency algorithms using publicly available model checkpoints and benchmark datasets. It does not collect private data, involve human subjects, or release new model checkpoints or datasets, and the experiments follow responsible research practices.

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: As a plug-and-play inference-efficiency method requiring neither an auxiliary draft model nor additional training, SPIRIT can reduce the computational cost of LLM inference by improving decoding efficiency. Since lower inference cost can also make existing LLMs easier to deploy at scale, we clarify that SPIRIT preserves the target model's output behavior and that model-level safeguards remain governed by the original model and deployment pipeline.

    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: The paper does not release a new model checkpoint or dataset. SPIRIT is an inference-time efficiency method evaluated on existing public checkpoints and benchmarks. Since full-model verification preserves the target model's output distribution, deployment-time safeguards remain those of the underlying model and serving environment.

    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: The paper identifies and cites the model checkpoints, benchmark datasets, and baseline methods used in the experiments. All datasets and model checkpoints are accessed from their public release channels and used according to their respective licenses and access terms, with model checkpoints, dataset sources, splits, and evaluation settings documented in Appendix C.

    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: [N/A]

    Justification: The paper does not release new public assets.

    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, user studies, or research with 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-subject research or collection of personal data; therefore IRB approval or equivalent review is not applicable.

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: LLMs are the object of study in this work. The paper describes the evaluated target checkpoints, self-speculative decoding setup, prompting, generation settings, and evaluation protocol in the method section and Appendix C.

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.