SELF-SPECULATION FOR FASTER REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning language models generate long chains of thought (CoT) before responding, and for long-form outputs such as code, generating the final response accounts for a substantial share of end-to-end latency. Speculative decoding can accelerate this phase, but existing drafters either copy text the model has already written, as in prompt lookup, or use a cheaper approximation of the model, like a small draft model, to guess a few tokens ahead. Reasoning models expose an additional source of speculation: complete responses produced by the model from a partial CoT, which increasingly overlap with the final response as reasoning progresses. We introduce SSR, a training-free self-speculative decoding method that uses a model's own partial-CoT response to accelerate generation of its final response, deriving both draft and verifier from the same model at different reasoning budgets. The draft is generated concurrently with the remaining reasoning, hiding most of its cost, and is reused through prefix verification and suffix decoding to speculate the final response. SSR accelerates the response rather than reasoning and requires no auxiliary models. Across three model families (Gemma 4, Qwen3, and GPT-OSS), SSR achieves 1.23–$1.55\times$ speedups over standard decoding on long-form code generation (ClassEval) with comparable task accuracy; the draft alone yields 1.22–$1.27\times$, and on top of suffix decoding with the same lookup mechanism, SSR adds a further 6–8% on Gemma and Qwen. We also analyze when SSR helps, including the effect of draft timing and length and where the reused text comes from. Code will be released upon publication.

1 INTRODUCTION

Reasoning language models are pushing LLMs beyond short-form assistance and into tasks that require sustained problem solving, including planning, coding, and decision making. These models achieve stronger performance by generating long reasoning traces that can span thousands of tokens before producing a final response. While this improves answer quality, it also increases end-to-end generation latency. For tasks with long outputs such as code, generating the final response itself contributes substantially to this latency, even after reasoning ends.

Speculative decoding is a natural fit for this setting. In standard speculative decoding, a draft model proposes multiple future tokens, which are then verified in parallel by the target model (Leviathan et al., 2023; Chen et al., 2023). When the draft is accurate, the verifier can accept several tokens at once and reduce the number of sequential decoding steps. The main challenge is constructing a draft that is both cheap to produce and accurate enough that the target accepts most of its tokens.

Previous work has approached this in different ways. A fully independent draft model is the natural baseline, but it must be trained separately, kept in memory alongside the target model, and carefully aligned with the target model's output distribution. Medusa (Cai et al., 2024) and EAGLE (Li et al., 2024) reduce this overhead by training lightweight draft heads or small modules that reuse parts of the target model, but still require training and architectural modifications. Self-speculative methods avoid auxiliary parameters by drafting from an approximation of the target model's own computation, using early layers (Elhoushi et al., 2024; Liu et al., 2024), sparse attention (Yue et al., 2026; Zhao et al., 2025), or quantization (Tiwari et al., 2025). At the other end of the spectrum, model-free methods (Saxena, 2023; Oliaro et al., 2024; Hu et al., 2025) copy text from the prompt and prior generations using n-gram lookup or suffix structures. These approaches differ in drafting cost and acceptance rate, but each guesses a few tokens ahead, either by imitating the model or by reusing text

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

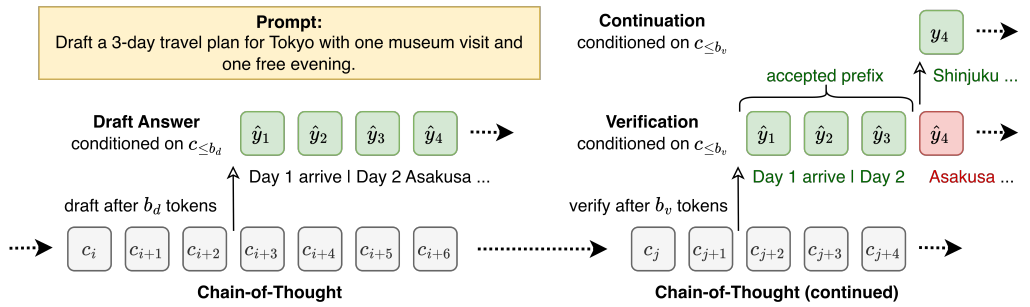

Figure 2: Illustration of concurrent generation in SSR. When reasoning reaches the draft budget, the same model drafts a response from the partial CoT while reasoning continues in parallel. Once reasoning completes, the draft is reused to speculate the final response.

it has produced. Model-free lookup is particularly effective for reasoning models, whose traces often contain fragments of the final response. That said, it can only reuse what has already been written.

Reasoning language models expose an additional source for speculation: the chain-of-thought itself. Work on test-time scaling has shown that CoT length can be controlled through budget forcing (Muennighoff et al., 2025), and that increasing the reasoning budget generally improves answer quality, at least up to a point. This suggests that partial CoTs are not arbitrary incomplete states, and that they can induce meaningful intermediate responses. Responses sampled early in the CoT might be incomplete or inconsistent, but as reasoning progresses they become increasingly similar in content to the final response (Figure 1). Unlike text already in the context, such a response is a complete draft produced by the full model, and it can contain content that the reasoning trace has planned but not yet written out. To the best of our knowledge, partial-CoT responses have not previously been used as a source of drafts for speculative decoding.

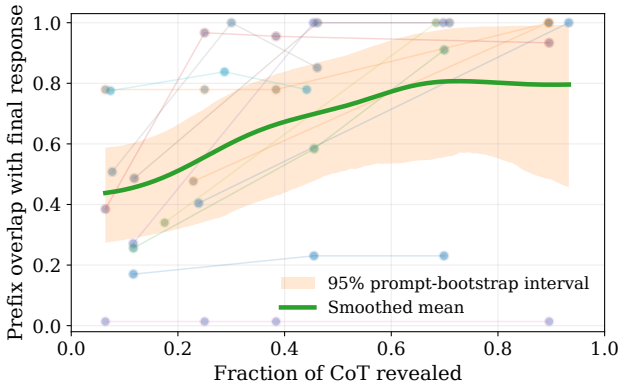

Figure 1: Drafts converge toward the final response as reasoning progresses. Thin lines: individual ClassEval prompts (Gemma-4-E4B-it); thick line: smoothed mean with 95% bootstrap interval.

We take advantage of this property to propose SSR: Self-Speculation for Reasoning Models. When reasoning reaches a draft budget, SSR samples a response conditioned on the partial CoT, while the same model continues reasoning toward its full budget. Drafting and continued reasoning proceed concurrently, so the drafting overhead is largely hidden behind ongoing CoT generation (Figure 2). Once reasoning completes, SSR uses the draft as an additional source for speculating the final response, alongside the prompt, reasoning trace, and response so far. SSR aims to use as much of the draft as possible: the verifier first accepts the longest draft prefix supported by the full-CoT distribution, and suffix decoding then proposes matching spans from the draft beyond the first rejection. This second stage matters because two responses may disagree early on minor details like variable names, while still agreeing on long spans later. Because prefix verification is not exact after the first rejection, we report task quality separately from latency.

SSR is orthogonal to the choice of drafting mechanism. Existing self-speculative methods approximate the model's computation to predict the next few tokens, and accelerate every decoding step, including reasoning. SSR instead runs the full model from an earlier reasoning state to draft the entire response, and accelerates only response generation. It changes what text is available for speculation rather than how tokens are proposed, and can be layered on top of these methods.

We implement SSR in vLLM and evaluate it on ClassEval (Du et al., 2023), a class-level code generation benchmark with long responses, across three model families (Gemma-4-E4B, Qwen3-4B-Thinking, and GPT-OSS-20B), and on HumanEval (Chen et al., 2021), a shorter-response coding benchmark, with Gemma. SSR achieves 1.23–$1.55\times$ speedups over standard decoding with comparable task accuracy, and the draft alone, used as the only source for speculation, yields 1.22–$1.27\times$. Much of SSR's speedup comes from reusing the prompt, reasoning trace, and response generated so far. To isolate the draft's contribution, we compare against suffix decoding over exactly these sources, using the same lookup mechanism and identical reasoning traces in both arms: SSR improves speedup by a further 8.3% on Gemma and 6.1% on Qwen. This gain depends on how far into the reasoning the draft begins and how long it runs: with the start fixed, a shorter draft yields no resolvable gain, and source attribution shows that much of each response already appears in the prompt, so the draft adds value mainly when it extends beyond that content.

In summary, our contributions are:

- We identify partial-CoT responses as a draft source for speculative decoding in reasoning models, and show that they increasingly resemble the final response as reasoning progresses.
- We propose SSR, a training-free method that drafts responses concurrently with ongoing reasoning and reuses them through prefix verification and suffix decoding, implemented in vLLM without changes to model weights.
- We evaluate SSR across three model families, isolating the draft's contribution with controlled comparisons and analyzing how draft timing and length affect it.

2 Related Work

2.1 Speculative decoding

The key driver of generation latency for autoregressive language models is the decode step, which is repeated across a large number of tokens. Speculative decoding reduces this cost by using a cheaper draft model to propose multiple tokens in advance, which are then verified in parallel by the target model (Leviathan et al., 2023; Chen et al., 2023). This line of work was developed as a lossless acceleration. Draft proposals need not come from a model: deterministic proposals, such as n-grams copied from the context, can also be verified exactly by treating them as a point-mass distribution. More recent work relaxes verification itself (Zhong et al., 2025; Bachmann et al., 2025), trading exactness for higher acceptance rates.

Draft model construction. The speedup speculative decoding provides is a function of both drafting overhead and acceptance rate. For speculative decoding to accelerate generation, the draft model must be cheap to run and close enough to the target model to maintain a high acceptance rate. In the open-source community, it is common to use a smaller, distilled variant of a flagship model as the draft model. But because the draft model also occupies significant memory, many popular approaches share parameters and computation between draft and target to minimize this overhead. Li et al. (2024) predicts penultimate-layer hidden states for future positions using lightweight feature predictors, then uses the last layer of the target model to predict the corresponding tokens in parallel. Cai et al. (2024) adds additional language modeling heads to decode future positions from the same hidden state. These approaches substantially reduce the footprint of drafting, but still require target-model-specific training and architectural modifications.

Self-speculative decoding. Another line of work eliminates the draft model entirely, drafting with a lossy approximation of the target model — via early exiting, sparsity, or quantization — and verifying with full inference. Elhoushi et al. (2024) trains intermediate layers to draft tokens for full-model verification, and Liu et al. (2024) adds an adaptive controller to decide when to exit and speculate. Quantization- and sparsity-based approaches can avoid training entirely. Tiwari et al. (2025) drafts using a hierarchical quantized KV cache, and Yue et al. (2026) and Zhao et al. (2025) draft using sparse attention; Zhao et al. (2025) targets long reasoning traces specifically, using attention scores to select a small set of important tokens. These methods approximate the model's computation to predict the next few tokens. SSR instead runs the full model from an earlier point in its reasoning, and could be combined with these methods, for example to draft or reason more cheaply.

Model-free methods. Model-free approaches propose spans of text with negligible overhead by copying text that already exists, and differ mainly in where they look for text and how they match it. N-gram speculative decoding (Saxena, 2023) proposes fixed-length n-grams drawn from the prompt and previously generated tokens, working well in domains with high lexical overlap. Suffix decoding and related work (Oliaro et al., 2024; Hu et al., 2025) extend this by building a suffix tree or automaton over this text, and optionally over previous requests, enabling longer, variable-length span proposals based on the currently matched suffix of the generated tokens. Retrieval-based methods such as REST (He et al., 2024) instead draw candidates from an external datastore, while Lookahead decoding (Fu et al., 2024) generates its own pool of n-grams through parallel Jacobi iterations. These methods are fast and particularly effective for reasoning models, whose reasoning traces often contain fragments of the final response, but they can only repeat text that has already been written. SSR is complementary to these mechanisms: rather than changing how text is matched, it adds a new source for them to search, a response drafted from a partial reasoning trace.

Speculation for reasoning models. Several recent methods accelerate reasoning models by speculating at the level of reasoning steps. SpecReason (Pan et al., 2025) and Reward-Guided Speculative Decoding (Liao et al., 2025) let a smaller model generate reasoning steps that the target model or a reward model accepts or corrects, Speculative Thinking (Yang et al., 2025b) uses a larger model to guide a smaller model's reasoning, and Lookahead Reasoning (Fu et al., 2025) layers step-level speculation on top of token-level speculative decoding. These methods speed up the reasoning itself, usually with a second model and a looser, meaning-based check of each step. SSR is complementary: it uses the same model, targets the final response rather than the reasoning trace, and draws its drafts from an earlier point in the model's own reasoning.

2.2 Inference-time compute and budgeted reasoning

Recent work has shown a clear tradeoff between inference-time compute and answer quality in reasoning language models, driving methods to control reasoning length at test time. Budget forcing controls the number of thinking tokens by forcing early termination or extending reasoning, for example by appending continuation tokens, with answer quality generally improving as the budget grows (Muennighoff et al., 2025). A parallel line of work studies when outputs stabilize during a reasoning trace, using stability or confidence signals to stop reasoning early (Liu and Wang, 2025; Yang et al., 2025a). More broadly, work on test-time compute scaling examines how to allocate additional inference compute to improve answer quality (Snell et al., 2024; Welleck et al., 2024). These methods primarily treat intermediate-budget outputs as cheaper final predictions or early-stop endpoints. SSR instead continues reasoning to the full budget and reuses intermediate outputs as drafts, so that computation spent on them accelerates the final response.

3 Preliminaries

3.1 Reasoning language models

We focus on applications where a language model generates a reasoning trace before producing a final response. Given a query q, the model first samples a chain-of-thought r token by token until it emits a model-specific end-of-thinking delimiter, and then samples the final response a conditioned on both the query and the reasoning trace until an end-of-sequence token is reached:

$$p_\theta(a \mid q, r) = \prod_{t=1}^{|a|} p_\theta(a_t \mid q, r, a_{<t}).$$

Reasoning can also be interrupted. In budget forcing (Muennighoff et al., 2025), the delimiter is appended after b reasoning tokens, and the model responds conditioned on the partial trace $r_{\leq b}$. We refer to $p_\theta(a \mid q, r_{\leq b})$ as the partial-CoT response distribution.

3.2 Speculative decoding

In speculative decoding (Leviathan et al., 2023; Chen et al., 2023), a draft distribution d proposes a sequence of tokens $\hat{a}_1, \ldots, \hat{a}_n$, which are then verified in parallel by the target distribution p. Tokens

are accepted sequentially until the first rejection; we refer to the accepted tokens as the accepted prefix. A draft token $\hat{a}_t$ is accepted with probability

$$\alpha_t = \min\left(1, \frac{p(\hat{a}_t \mid q, r, a_{<t})}{d(\hat{a}_t \mid q, r, a_{<t})}\right),$$

and upon rejection, a token is sampled from the residual distribution

$$a_t \sim \frac{\max(p - d, 0)}{\sum_v \max(p(v) - d(v), 0)}.$$

Together, acceptance and residual sampling guarantee that the final output distribution is identical to that of the target model p, regardless of draft quality. Because many draft tokens can be verified in a single forward pass, a high acceptance rate reduces the number of sequential decoding steps.

3.3 Model-free speculation over token sources

Model-free methods draft tokens by copying text that already exists, instead of sampling from a draft model. We call the text they copy from the *source*. At each step, the method looks for the last few generated tokens in the source and, if it finds them, proposes the tokens that came next. N-gram speculative decoding (Saxena, 2023) matches a fixed number of tokens against the prompt and the output so far. Suffix decoding (Oliaro et al., 2024) stores the source in a suffix tree, which lets it find longer matches and propose longer continuations. The source can cover only the current request or also include earlier requests; we use only the current request throughout. Copied tokens have no draft probabilities, but they can still be verified exactly: a proposed token $\hat{a}_t$ is accepted with probability $p(\hat{a}_t \mid \cdot)$, and if it is rejected, a replacement is sampled from p with $\hat{a}_t$ excluded. We use this exact rule for all suffix-decoding proposals. In these terms, SSR adds a new source: a response drafted from a partial reasoning trace.

4 Method

SSR accelerates the final response of a reasoning model in two stages (Figure 2). First, when reasoning reaches a draft budget, the same model drafts a response from the partial chain-of-thought while reasoning continues concurrently. Second, once reasoning completes, SSR generates the final response using the draft as an additional source for speculation, alongside the prompt, the completed reasoning trace, and the response generated so far. Algorithm 1 gives the full procedure.

4.1 Drafting from a partial chain-of-thought

We instantiate speculative decoding using a single reasoning model evaluated at two reasoning budgets. When the reasoning trace reaches the draft budget b_d, SSR samples a draft response $\hat{a} \sim p_\theta(\cdot \mid q, r_{\leq b_d})$ of up to m tokens, while the model continues reasoning toward completion. Conditioning on the partial trace requires appending the model's end-of-thinking delimiter, as in budget forcing. The key observation is that draft generation and continued reasoning can proceed concurrently, so the overhead of drafting is largely hidden behind ongoing reasoning.

For the draft to reach its full length before reasoning ends, the total CoT length must exceed the draft budget plus the maximum draft length,

$$|r| > b_d + m.$$

If reasoning ends first, SSR stops drafting and uses the portion generated so far, so drafting never delays the final response. Beyond this, the choice of b_d involves a trade-off. Drafting early leaves more time to generate a long draft, but the partial trace may not yet have converged toward the final response; drafting late produces a draft that more closely resembles the final response (Figure 1), but leaves less time to generate it. If reasoning ends before the draft budget is reached, no draft is produced and SSR reduces to speculation over the remaining sources.

4.2 Using the draft

Once reasoning completes with trace r, SSR uses the draft in two ways.

Algorithm 1 SSR: Self-Speculation from Partial Reasoning

Input: Prompt q, draft budget b_d, maximum draft length m, model p_θ
Output: Final response a

Stage 1: Concurrent reasoning and drafting

$r \leftarrow \emptyset,\ \hat{a} \leftarrow \emptyset$
while reasoning has not ended **do**
  Sample $z \sim p_\theta(\cdot \mid q, r)$ and $r \leftarrow \mathrm{concat}(r, z)$
  if $|r| = b_d$ **then**
    Start drafting $\hat{a} \sim p_\theta(\cdot \mid q, r_{\le b_d})$, up to m tokens, in parallel ‖
  end if
end while

Stage 2: Prefix verification

$k \leftarrow \textsc{VerifyDraft}(\hat{a};\ q, r_{\le b_d}, r)$ ▷ $k = 0$ if no draft

Stage 3: Continuation with suffix decoding

Source $\mathcal{S} \leftarrow \{q,\ r,\ \hat{a}\}$, extended with the response as it is generated
Generate $a_{\mathrm{cont}} \sim p_\theta(\cdot \mid q, r, \hat{a}_{1:k})$ using suffix decoding over $\mathcal{S}$
return $\mathrm{concat}(\hat{a}_{1:k}, a_{\mathrm{cont}})$

Prefix verification. SSR first verifies the draft against the full-CoT distribution, treating $p_\theta(\cdot \mid q, r_{\le b_d})$ as the draft distribution and $p_\theta(\cdot \mid q, r)$ as the target distribution. Because all draft tokens are available at once, the entire draft can be verified in a single forward pass, and a long prefix can be accepted at once when the draft closely matches the final response. We write $k = \textsc{VerifyDraft}(\hat{a};\ q, r_{\le b_d}, r)$ for the length of the accepted prefix.

Suffix decoding. Generation then continues from the accepted prefix under the full-CoT distribution, using two lookups at each step. In our implementation, the prompt, reasoning trace, and response are indexed by a suffix tree, while the draft is searched by a longest-match lookup whose candidate is used when it is longer than the suffix tree's. Proposed tokens are verified with the exact rule described in Section 3. This stage recovers useful draft content beyond the first rejection. Two responses might disagree on minor details like variable names or phrasing early on, while still agreeing on long spans later in the text; prefix verification alone would discard these spans, whereas suffix decoding can still reuse them (Figure 3). Because the draft is one source among several, SSR benefits from it exactly when it contains text that the prompt, reasoning trace, and response so far do not.

Exactness. Suffix-decoding proposals are verified exactly. Prefix verification uses the standard acceptance ratio, but after the first rejection our implementation resumes ordinary decoding from the accepted prefix, sampling the replacement token from $p_\theta(\cdot \mid q, r, \hat{a}_{1:k})$ rather than from the residual distribution. This affects at most one token per response; applying the residual correction would make SSR exact, and is a straightforward extension. We report task quality separately from latency.

4.3 Serving-time implementation

We implement SSR in vLLM by duplicating the original request inside the scheduler when the draft budget is reached. The child request shares the same prefix as the original reasoning request, so vLLM's prefix caching reuses the computation already performed for the prompt and partial CoT. The child appends the end-of-thinking delimiter and begins drafting, while the parent request continues reasoning. Verification and continuation are likewise issued as ordinary vLLM requests, coordinated by scheduler metadata, with the final output returned under the original request identifier. Draft, verification, and continuation requests clone the parent request's sampling parameters, including its random seed; they are separate requests with independently initialized generators.

This implementation requires minimal changes to the serving stack, but it is not compute-free: during the overlap window, the scheduler serves both parent and child requests, increasing the batch size and total compute. The speedup comes from hiding draft latency behind ongoing reasoning, not from

Schematic

Mechanism	Accepted output structure
Prefix acceptance	**prefix prefix prefix** **rejected rejected**
Prefix + continuation	**prefix prefix prefix** cont cont
Prefix + suffix reuse	**prefix prefix prefix** cont cont **suffix suffix** cont

accepted prefix; **rejected draft**; regular decoding; **accepted suffix**.

Example

```
from datetime import datetime, timedelta
class CalendarUtil:  ...
def remove_event(self, event):  ...
>> calendar.events = [{'date':  datetime(2023, 1, 1, 0, 0),
...    'start_time':  datetime ( 2023, 1, 1, 0, 0 ),
...    'end _time ':  datetime(2023, 1, 1,  1, 0 ),
...    'description':  'New Year'}]
```

Figure 3: Top: schematic illustration of prefix verification and suffix reuse. Bottom: compressed real code-generation example showing the same structure.

reducing FLOPs. Appendix A gives details, and Appendix F describes a multi-stage variant that refreshes the draft at several reasoning budgets.

5 Experiments

We evaluate whether the partial-CoT draft accelerates final-answer generation beyond training-free reuse, where the reused text comes from, and how the benefit depends on when drafting starts.

5.1 Experimental setup

Models and benchmarks. We evaluate three reasoning models from different families: Gemma-4-E4B-it (Google DeepMind, 2026), Qwen3-4B-Thinking-2507, and GPT-OSS-20B (medium reasoning effort). Our main benchmark is ClassEval (Du et al., 2023), which asks models to implement complete Python classes, producing long responses. We use all 100 tasks for Gemma and Qwen, and 98 for GPT-OSS after excluding two tasks used for configuration. We additionally evaluate Gemma on HumanEval (Chen et al., 2021), whose responses are short. Each model uses its native end-of-thinking delimiter and recommended sampling settings (Appendix B).

SSR configuration. Reasoning is capped at 2000 tokens, which binds for 95% of Qwen and 31% of Gemma tasks. For Gemma and Qwen, drafting starts after 1000 reasoning tokens, with drafts of up to 1000 tokens; GPT-OSS reasons briefly (median 427 tokens; Appendix C), so it drafts after 200 tokens for up to 200 tokens. Suffix decoding (Oliaro et al., 2024) uses tree depth 4 and up to 16 speculative tokens per step, with its cross-request cache disabled. Section 5.4 reports all drafting configurations we evaluated.

Compared methods. *Standard decoding* uses no speculation. *Suffix decoding* speculates from the prompt, the completed reasoning trace, and the response so far; it is the controlled baseline, with access to all text available to SSR except the draft. *SSR* adds the partial-CoT draft as a further source (Section 4), so the difference between SSR and suffix decoding measures the draft's contribution. *Draft only* uses the draft as the sole source. Other model-free lookups, including vLLM's n-gram decoding, are compared in Appendix E.

Metrics. For each task, we compare against a same-token baseline that generates exactly as many tokens with standard decoding. We report mean end-to-end speedup over tasks, from prompt submission to the final token, so drafting cost is included. Our primary metric is the paired difference between SSR and suffix decoding with a 95% bootstrap confidence interval over tasks; we check per task whether both arms produce identical reasoning traces, and compare methods only within a

Table 1: Mean end-to-end speedup on ClassEval over a same-token standard-decoding baseline. The draft gain is the paired per-task difference between SSR and suffix decoding, with a 95% bootstrap confidence interval. Draft only uses the draft as the sole speculation source and was measured in separate runs, each against its own same-token baseline.

Model	Start / length	n	Draft only	Suffix	SSR	Draft gain [95% CI]
Gemma-4-E4B-it	1000 / 1000	100	$1.215\times$	$1.423\times$	$1.541\times$	$+0.118$ [0.066, 0.173]
Qwen3-4B-Thinking	1000 / 1000	100	$1.272\times$	$1.164\times$	$1.234\times$	$+0.071$ [0.059, 0.083]
GPT-OSS-20B	200 / 200	98	$1.259\times$	$1.534\times$	$1.553\times$	$+0.019$ [-0.010, 0.048]

Table 2: Pass@1 on unit tests, over completed and evaluable responses. Standard decoding uses no speculation. Δ is SSR minus suffix decoding in percentage points.

Model	Benchmark	Standard	Suffix	SSR	Δ (pp)	McNemar p
Gemma-4-E4B-it	ClassEval	38/86	38/86	36/86	-2.3	1.0
Qwen3-4B-Thinking	ClassEval	31/86	28/86	27/86	-1.2	1.0
Gemma-4-E4B-it	HumanEval	154/164	138/147	140/146	$+2.0$	1.0

run. Because prefix verification is not exact after the first rejection (Section 4), we report pass@1 on the benchmarks' unit tests. Latency experiments run one request at a time per GPU worker on NVIDIA RTX A5000 GPUs (A6000 for HumanEval), with GPT-OSS using tensor parallelism across two GPUs.

5.2 Does the draft help beyond suffix decoding?

Table 1 summarizes the results. End to end, SSR is 1.23–$1.55\times$ faster than standard decoding, and the draft alone, used as the only speculation source, gives 1.22–$1.27\times$. Suffix decoding over the prompt, reasoning trace, and response already provides a substantial speedup; adding the draft improves on it by $+0.118\times$ (8.3%) on Gemma and $+0.071\times$ (6.1%) on Qwen, with confidence intervals excluding zero and identical reasoning traces in both arms on every task. Only 72 of 100 Gemma tasks reason long enough to start a draft (97 of 100 on Qwen); on those 72, the gain is $+0.149\times$ [0.077, 0.224]. GPT-OSS reasons too briefly to leave much room for drafting and shows no clear gain; its two arms also produced different reasoning traces on 96 of 98 tasks, so its paired comparison is noisier.

Since SSR does not speed up reasoning, its effect on the response alone is larger. Measured from the end of reasoning, SSR produces the response $1.23\times$ faster than suffix decoding on Gemma and $1.28\times$ faster on Qwen (total tokens over total time; per-task averages are $1.26\times$ and $2.25\times$), with nearly identical response lengths.

Appendix E reports comparisons with other model-free lookup mechanisms, including vLLM's stock n-gram decoding and a controlled check in which both arms use the same longest-match lookup; the latter also shows a gain from adding the draft ($+0.161\times$ on Gemma).

Other benchmarks. On HumanEval, where responses are short and reasoning dominates latency, SSR still improves speedup over suffix decoding on Gemma, though by less: from $1.112\times$ to $1.141\times$ (paired gain $+0.029\times$ [0.002, 0.057], $n = 164$). Only 79 of 164 tasks reach the draft start, consistent with shorter reasoning traces leaving less room to draft.

Task quality. Table 2 reports pass@1. Suffix-decoding proposals are verified exactly, and prefix verification differs from exact sampling in at most one token per response, so we expect quality to match standard decoding up to sampling variation. Pass rates are comparable across methods, and paired differences between the suffix-decoding and SSR arms are not statistically significant.

5.3 Where does the reused text come from?

To understand these results, we attribute each accepted speculative token to the sources containing it, at the default configuration (Table 6, Appendix D). Two patterns stand out. First, the prompt dominates: ClassEval prompts include a class skeleton with method signatures and docstrings, and

Table 3: Draft timing and length both matter. Paired draft gain (SSR minus suffix decoding) on ClassEval, with 95% bootstrap CIs; bold CIs exclude zero. *Drafted*: tasks that reached the draft start.

	Gemma-4-E4B-it		Qwen3-4B-Thinking	
Start / length	Drafted	Draft gain	Drafted	Draft gain
500 / 500 (default)	96/96	$\mathbf{+0.062}$ [0.006, 0.124]	100/100	$+0.003$ [-0.011, 0.015]
1000 / 500	72/100	$+0.005$ [-0.035, 0.045]	–	–
1250 / 500	55/55	$+0.026$ [-0.003, 0.054]	97/97	$\mathbf{+0.047}$ [0.033, 0.061]
1000 / 1000	72/100	$\mathbf{+0.118}$ [0.066, 0.173]	97/100	$\mathbf{+0.071}$ [0.059, 0.083]

55–67% of each model's final response appears verbatim in the prompt, so much of what a draft could supply is already available. Second, the share of accepted tokens found only in the draft is small, from 0.8% for GPT-OSS to 12.3% for Qwen. GPT-OSS illustrates why: its reasoning is largely abstract, with only 3.7% of final-response tokens appearing in the reasoning trace, the regime where a draft should help most. However, its drafts begin early and are short, and inspection shows they spend most of their budget reproducing the class definition and docstrings already in the prompt, ending before the method bodies where the draft's unique content would lie. For Qwen, the draft contributes the largest unique share but no latency gain here; later and longer drafts recover a gain (Section 5.4).

5.4 When should drafting start, and how long should it run?

Figure 1 shows that drafts grow more similar to the final response as reasoning progresses, mostly in early-to-mid reasoning. Along fixed reasoning trajectories for 12 ClassEval tasks, exact prefix overlap with the final response increased for 10 of 12 tasks between 128 and 500 reasoning tokens, with none decreasing; the median within-task Spearman correlation between reasoning progress and overlap was 0.87. With the attribution results, this suggests our default configuration drafts too early and too briefly: the draft is less mature, and its budget is spent on content the prompt already provides.

Table 3 tests this. Starting later helps when enough reasoning remains: on Qwen, moving the start from 500 to 1250 tokens turns a negligible gain into $+0.047\times$. Draft length also matters: on Gemma, with the start fixed at 1000 tokens, a 500-token draft gives no resolvable gain, whereas a 1000-token draft gives $+0.118\times$. These results suggest a simple heuristic: draft roughly the typical response length, starting as late as leaves enough reasoning to generate it. We identified this heuristic on the same tasks we evaluate on; the 1000/1000 configuration was selected on Gemma and applied to Qwen unchanged.

6 Limitations and Future Work

SSR is most effective when answer length is comparable to CoT length; when reasoning dominates generation time, gains are limited since CoT generation is not accelerated. In high-entropy tasks, drafts may be semantically similar to the final response but share little lexical overlap, limiting prefix and suffix reuse. Natural directions for future work include extending SSR to accelerate CoT generation itself and fine-tuning models to commit to response structure earlier in the CoT.

7 Conclusion

We introduce SSR, a training-free self-speculative decoding method for reasoning language models that uses the chain-of-thought as a source of speculation. By conditioning the same model at two reasoning budgets, SSR derives both draft and verifier without auxiliary models or target-model-specific modifications, hiding draft generation overhead behind concurrent reasoning. Combined with suffix decoding to recover useful draft content beyond the accepted prefix, SSR reduces end-to-end latency on tasks like long-form code generation with comparable task quality, and a draft started late enough and run long enough adds a consistent gain over suffix decoding. We hope this work encourages exploration of reasoning structure as a resource for efficient generation.

AI USE STATEMENT

Generative AI tools were used to assist with research methodology and experiment design, software implementation and debugging, result interpretation, literature search and summarization, and drafting, editing, and formatting portions of the manuscript. All AI-assisted outputs were reviewed and verified by the authors, who take responsibility for the final content of this work, including text, claims, code, and experimental artifacts produced with the aid of generative AI.

REFERENCES

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali K. Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=mtSSFiqW6y.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 5209–5235. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *CoRR*, abs/2302.01318, 2023. doi: 10.48550/ARXIV.2302.01318. URL https://doi.org/10.48550/arXiv.2302.01318.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura 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, Mo Bavarian, Clemens Winter, Phil Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. 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*, abs/2107.03374, 2021. URL https://api.semanticscholar.org/CorpusID:235755472.

Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation. *ArXiv*, abs/2308.01861, 2023. URL https://api.semanticscholar.org/CorpusID:260439062.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.681. URL https://doi.org/10.18653/v1/2024.acl-long.681.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 14060–14079. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

Yichao Fu, Rui Ge, Zelei Shao, Zhijie Deng, and Hao Zhang. Scaling speculative decoding with lookahead reasoning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/fb65f4aed3027871f349dbc91cd27ae4-Abstract-Conference.html.

Google DeepMind. Gemma 4. https://deepmind.google/models/gemma/gemma-4/, April 2026. Accessed: 2026-05-06.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: retrieval-based speculative decoding. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 1582–1595. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.NAACL-LONG.88. URL https://doi.org/10.18653/v1/2024.naacl-long.88.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 12187–12204. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.595/.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pages 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 28935–28948. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/li24bt.html.

Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. Reward-guided speculative decoding for efficient LLM reasoning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/liao25f.html.

Jiahao Liu, Qifan Wang, Jingang Wang, and Xunliang Cai. Speculative decoding via early-exiting for faster LLM inference with thompson sampling control mechanism. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pages 3027–3043. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.179. URL https://doi.org/10.18653/v1/2024.findings-acl.179.

Xin Liu and Lu Wang. Answer convergence as a signal for early stopping in reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 17896–17907. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.904. URL https://doi.org/10.18653/v1/2025.emnlp-main.904.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1025. URL https://doi.org/10.18653/v1/2025.emnlp-main.1025.

Gabriele Oliaro, Zhihao Jia, Daniel F. Campos, and Aurick Qiao. Suffixdecoding: A model-free approach to speeding up large language model inference. *CoRR*, abs/2411.04975, 2024. doi: 10.48550/ARXIV.2411.04975. URL https://doi.org/10.48550/arXiv.2411.04975.

Rui Pan, Yinwei Dai, Zhihao Zhang, Gabriele Oliaro, Zhihao Jia, and Ravi Netravali. Specreason: Fast and accurate inference-time compute via speculative reasoning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/12c45a68e8433b21b91cd47731387fa4-Abstract-Conference.html.

Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *CoRR*, abs/2408.03314, 2024. doi: 10.48550/ARXIV.2408.03314. URL https://doi.org/10.48550/arXiv.2408.03314.

Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Richard Charles Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Quantspec: Self-speculative decoding with hierarchical quantized KV cache. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/tiwari25b.html.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaïd Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=eskQMcIbMS.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping Wang. Dynamic early exit in reasoning models. *CoRR*, abs/2504.15895, 2025a. doi: 10.48550/ARXIV.2504.15895. URL https://doi.org/10.48550/arXiv.2504.15895.

Wang Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. *CoRR*, abs/2504.12329, 2025b. doi: 10.48550/ARXIV.2504.12329. URL https://doi.org/10.48550/arXiv.2504.12329.

Yikang Yue, Yuqi Xue, and Jian Huang. Specattn: Co-designing sparse attention with self-speculative decoding. *CoRR*, abs/2602.07223, 2026. doi: 10.48550/ARXIV.2602.07223. URL https://doi.org/10.48550/arXiv.2602.07223.

Yilong Zhao, Jiaming Tang, Kan Zhu, Zihao Ye, Chi-Chih Chang, Chaofan Lin, Jongseok Park, Guangxuan Xiao, Mohamed S. Abdelfattah, Mingyu Gao, Baris Kasikci, Song Han, and Ion Stoica. Accelerating large-scale reasoning model inference with sparse self-speculative decoding, 2025. URL https://arxiv.org/abs/2512.01278.

Meiyu Zhong, Noel Teku, and Ravi Tandon. Speeding up speculative decoding via approximate verification. *CoRR*, abs/2502.04557, 2025. doi: 10.48550/ARXIV.2502.04557. URL https://doi.org/10.48550/arXiv.2502.04557.

A SERVING-TIME IMPLEMENTATION

When the reasoning request reaches the draft budget, the scheduler spawns a child DRAFTING request initialized with the same token prefix, followed by the model-specific end-of-thinking tokens. The child generates up to a configured draft-token budget while the parent continues along the longer reasoning path.

In multi-drafting mode, the scheduler spawns draft children at fixed token increments, storing outputs by spawn point. When the parent finishes, outstanding draft children are removed and the latest usable draft is selected for verification.

Verification is implemented as a VERIFICATION request whose prompt is the final reasoning prefix followed by the selected draft tokens. Prompt log probabilities are compared against the draft's log probabilities and a prefix is accepted using the standard speculative decoding rule. A CONTINUATION request then generates remaining answer tokens from the accepted prefix, optionally using the draft as a suffix cache source. All stages are ordinary vLLM requests coordinated by scheduler metadata, with the final output returned under the original request id.

B INFERENCE HYPERPARAMETERS

All latency experiments run one request at a time per vLLM worker, in bfloat16. Each speedup is measured against a same-token standard-decoding baseline that generates exactly as many tokens as the accelerated request. Latency is measured end to end around generation, excluding model loading. Runs used eager mode; ClassEval used 0 warmup and 1 measured run, while Gemma HumanEval used 1 warmup and 3 measured runs.

Table 4: Per-model settings for the main experiments.

	Gemma-4-E4B-it	Qwen3-4B-Thinking-2507	GPT-OSS-20B
Temperature / top-p / top-k	0.6 / 1.0 / –	0.6 / 0.95 / 20	0 / 1 / 0
Seed	0	0	0
End-of-thinking delimiter	`<channel|>`	`</think>`	`<|end|>` (ID 200007)
Reasoning cap	2000	2000	2000
Draft start / max length	1000 / 1000	1000 / 1000	200 / 200
Tensor parallelism	1	1	2
GPU	RTX A5000	RTX A5000	RTX A5000

Suffix decoding uses tree depth 4, up to 16 speculative tokens per step, a maximum speculation factor of 4, and no minimum token probability, with its cross-request cache disabled. When the model emits the end-of-thinking delimiter before the reasoning cap, reasoning ends at that point. HumanEval runs used RTX A6000 GPUs.

C REASONING AND RESPONSE LENGTHS

Table 5: Reasoning and response lengths on ClassEval (tokens), with the fraction of tasks whose reasoning exceeds 1000 tokens and the fraction that reach the 2000-token reasoning cap.

Model	Median CoT	Median response	CoT $>$ 1000	CoT at cap
Gemma-4-E4B-it	1434	1211	72%	31%
Qwen3-4B-Thinking	2000	682	98%	95%
GPT-OSS-20B (medium)	427	751	14%	4%

GPT-OSS reasons briefly, so drafting later than 200 reasoning tokens would apply to few tasks; in a 20-task screen at high reasoning effort, only 3 of 20 reasoning traces reached 1000 tokens.

Table 6: Where accepted speculative tokens can be found, for SSR on ClassEval at the default configuration. Sources can overlap, so columns need not sum to 100%. *Draft only*: tokens found in the draft and no other source. *Resp. in prompt*: fraction of final-response tokens appearing verbatim in the prompt (runs of ≥8 tokens).

Model	Prompt	CoT	Response so far	Draft	Draft only	Resp. in prompt
Gemma-4-E4B-it	67.2%	36.7%	77.3%	69.5%	3.2%	54.6%
Qwen3-4B-Thinking	68.2%	30.7%	49.3%	80.2%	12.3%	64.6%
GPT-OSS-20B	84.3%	14.4%	61.7%	29.8%	0.8%	67.4%

Algorithm 2 Multi-Stage Self-Speculative Decoding from Partial Reasoning

Input: Prompt q, budgets $b_1 < b_2 < \cdots < b_T$, model p_θ, suffix cache C
Output: Generated response y, updated suffix cache C

Generate reasoning up to budget b_1 to obtain r_1
Start draft generation $\hat{a}_1 \sim p_\theta(\cdot \mid q, r_1)$ in parallel
for $i = 1, \ldots, T-1$ **do**
    Continue reasoning to budget b_{i+1} to obtain r_{i+1}
    Collect draft $\hat{a}_i$ and update suffix cache C
    $k_i \leftarrow \textsc{VerifyDraft}(\hat{a}_i;\ q, r_i, r_{i+1})$
    $\hat{a}_{i,1:k_i}$ is the longest prefix accepted under the higher-budget context r_{i+1}.
    if $i < T-1$ **then**
        Start next-stage draft generation

$$\hat{a}_{i+1} \sim p_\theta(\cdot \mid q, r_{i+1}, \hat{a}_{i,1:k_i})$$

    Reuse the accepted prefix rather than drafting from scratch.
    end if
end for
Sample continuation $a_{\text{cont}} \sim p_\theta(\cdot \mid q, r_T, \hat{a}_{T-1,1:k_{T-1}})$
    Use suffix decoding with cache C during continuation.
return $\text{concat}\left(r_T, \hat{a}_{T-1,1:k_{T-1}}, a_{\text{cont}}\right), C$

D Source Attribution Details

E Other Lookup Mechanisms

The comparisons in Section 5.2 hold the suffix tree fixed. In SSR, the draft is searched by a longest-match lookup rather than by the suffix tree, so part of the gain may reflect that lookup rather than the draft's content. In an earlier configuration where both arms use the same longest-match lookup over the prompt and reasoning trace (drafting after 500 reasoning tokens on Gemma, ClassEval, $n = 100$), adding the draft improves speedup from $1.600\times$ to $1.761\times$ (paired gain $+0.161\times$ [0.077, 0.248]).

For reference, we also report uncontrolled comparisons with other model-free methods, each measured against its own same-token baseline. A longest-match lookup over the prompt and reasoning trace alone reaches $1.545\times$ on Gemma (1000/1000 configuration, identical reasoning traces to the arms in Table 1, but a different lookup mechanism and host). vLLM's stock n-gram decoding applied to the final response reaches $1.738\times$ [1.661, 1.820] on Gemma and $1.213\times$ [1.190, 1.238] on Qwen; enabling the stock speculative runtime changed the sampled reasoning traces (28 of 100 matched on Gemma), so these are not controlled comparisons with SSR. Combining SSR's draft with these lookup mechanisms is future work.

F Multi-Stage Self-Speculative Decoding