EXTRACTIVE SPECULATIVE DECODING: LEMPEL-ZIV LIMITS AND HOW TO APPROACH THEM

Anonymous authors
Paper under double-blind review

ABSTRACT

Copying candidate continuations from previously observed text provides a lightweight, model-agnostic approach to speculative decoding, which we call *extractive speculation*. Its performance, however, is fundamentally limited by what is available to copy. We show that this limitation has a precise connection to Lempel–Ziv compression and derive a tight, sequence-dependent upper bound on the acceptance length of any extractive speculator. We compute the resulting *LZ bound* exactly on standard benchmarks and show that it falls below existing generative speculators. Hence, under standard benchmark settings, no extractive method can close the gap to generative speculation. We further show that substantially enlarging the available copy corpus raises this bound and can make extractive speculation effective. However, existing methods remain well below the *LZ bound* even in these settings. We therefore introduce a representation-space retrieval method that uses the target model's hidden states and adapts its similarity metric online using supervision derived from our theory. Our method consistently improves over existing extractive approaches and narrows the gap to the LZ bound, while substantial headroom remains.

1 INTRODUCTION

Auto-regressive decoding of large language models (LLMs) is often bounded by the memory-bandwidth as transferring model weights from memory for each generated token dominates the wall-clock time. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) exploits the idle arithmetic capacity by verifying multiple candidate continuations, called *drafts*, in parallel. While the original speculative decoding framework uses a smaller language model as the drafter, subsequent work has explored a wide range of strategies, including multi-token prediction heads (Cai et al., 2024; Samragh et al., 2025; Christopher et al., 2025; Chen et al., 2026), more efficient draft models (Li et al., 2024b; 2025; Liu et al., 2024), tree-based verification (Miao et al., 2024; Li et al., 2024a), and self-speculation (Zhang et al., 2024; Elhoushi et al., 2024; Fu et al., 2024; Liu et al., 2025). Among these approaches, a family of methods constructs drafts entirely by copying verbatim token sequences from an available source, typically the current context (Yang et al., 2023; Saxena, 2023; Dumitru et al., 2025; Lee et al., 2025), but also a global suffix tree accumulated across requests (Oliaro et al., 2025) or an external datastore (He et al., 2024; Quan et al., 2025). We refer to these methods as *extractive speculation*, in contrast to *generative speculation*, where drafts are generated rather than copied verbatim from another text.

Compared to generative speculation, extractive speculation offers three practical advantages. *(1) Low overhead*: draft construction reduces to retrieval or string matching rather than a forward pass through a neural drafter, reducing drafting latency from roughly 2 ms for EAGLE3 to 0.1 ms for suffix-tree lookup per token (Oliaro et al., 2025). *(2) Universality*: extractive speculation can be applied to any model without architectural modification or model-specific training. *(3) Natural scaling with available text*: the space of possible drafts grows as more context or external text becomes available. This property is increasingly relevant for long-context and multi-turn workloads like agentic settings that accumulate substantial interaction history and repeatedly access related code, documents, or tool outputs.

These advantages come with an inherent restriction. An extractive drafter can only propose continuations that already occur in its available corpus, whereas a generative drafter can, in principle,

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.

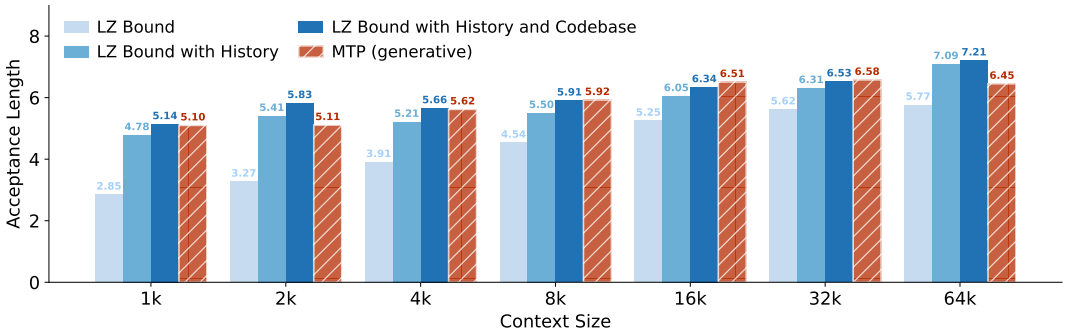


Figure 1: **Lempel-Ziv bound on extractive speculation vs. MTP as context grows.** We compute the LZ bound under three corpus settings: **LZ**: the current request and possibly the earlier turns in multi-turn sessions; **LZ + History:** additionally including ≈50K tokens from previous SWE-bench sessions, each corresponding to a different issue fix; and **LZ + History + Codebase:**, additionally including the full repository source code (≈35M tokens). The bound increases with the available context and additional information like history and codebase in line with context monotonicity. In the standard benchmark setting of LZ, the bound remains well below generative speculation, while adding history and the codebase closes most of this gap. Hence, extractive speculation can only be effective when the corpus is large. (SWE-bench, Qwen3.5-9B, $K{=}8$)

propose arbitrary continuations. This raises a fundamental question: *How much acceleration is possible when drafting is restricted to previously observed text?*

We show that this question has a direct connection to Lempel–Ziv (LZ) compression (Lempel & Ziv, 1976). Both LZ compression and extractive speculation exploit repeated substrings by copying them from previously observed text. Using this connection, we derive a tight upper bound on the acceptance length achievable by any extractive drafter. The resulting *LZ bound* can be computed exactly for a given sequence and copy corpus, allowing us to characterize the best extractive speculation performance possible independently of any particular retrieval algorithm.

Applying this bound to standard speculative-decoding benchmarks reveals a conclusive negative result. Under the conventional benchmark setup, the extractive upper bound itself falls below generative drafters like the target model's native multi-token prediction (MTP) module. Hence, under these benchmark settings, no extractive speculation algorithm can close the gap to generative speculation through better retrieval. The information in the available copy corpus is simply insufficient.

The theory also points to a natural remedy of enlarging the corpus from which drafts can be extracted. We show that the LZ bound is monotonic in the available copy corpus and empirically study this effect by progressively expanding the corpus available to the drafter. Incorporating additional interaction history and task-relevant sources significantly raises the extractive performance bound. With a sufficiently rich corpus, the resulting upper bound becomes comparable to, and in some settings exceeds, the acceptance lengths achieved by generative drafters.

Even after incorporating additional information, a second gap remains. When the enlarged corpus contains continuations that would make extractive speculation competitive, existing methods such as n-gram and suffix-based retrieval still remain far below the LZ bound. This second gap arises not from the availability of reusable text, but from the inability to retrieve the right continuation.

Motivated by this retrieval perspective, we develop a method operating in the model's representation space rather than relying on exact token matches. Moreover, the LZ analysis provides supervision for the optimal retrieval decisions. We use this signal to adapt the retrieval metric online while preserving the extractive nature of the drafter. Our method consistently improves over existing approaches and narrows the gap to the LZ bound. However, a substantial headroom remains.

In summary, we make the following contributions. i) We introduce the LZ bound, which precisely characterizes the maximum acceptance length achievable by extractive speculation for a given sequence and copy corpus. ii) We show that this bound lies below generative speculators under standard benchmark settings, establishing a fundamental limitation of extractive speculation when the

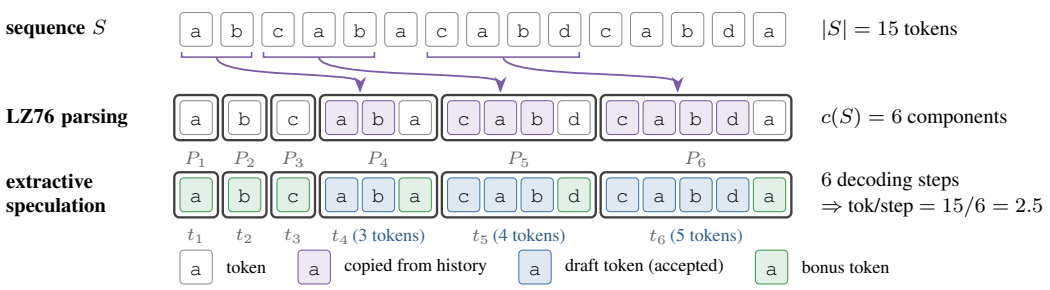


Figure 2: **Extractive speculation is equivalent to LZ parsing.** *Top:* a sequence of $M = 15$ tokens. *Middle:* its LZ parsing into $N = 6$ components, where each component consists of a copied substring (arrows indicating the source) followed by a new token. *Bottom:* the corresponding extractive speculation. Every component maps to one speculation step where the substring is the accepted draft (blue) and the following token is generated as the bonus token (green). The number of target-model calls is the number of LZ phrases with an average acceptance length of M/N.

available corpus is small. iii) We show theoretically and empirically that enlarging the copy corpus substantially improves this ceiling. iv) We demonstrate that existing extractive methods remain far from the LZ-bound and introduce a representation-space retrieval method with online adaptation that moves closer to it. More broadly, the LZ bound provides a guiding principle for extractive speculation as it identifies when extractive speculation is viable, quantifies the value of the available corpus, and provides a direct signal for evaluating and improving retrieval algorithms.

2 Preliminaries on Lempel-Ziv Complexity

Lempel–Ziv complexity characterizes the repetitive structure of a finite string by measuring how efficiently it can be constructed through copying from earlier positions in the string (Lempel & Ziv, 1976). Intuitively, a string has low complexity when long portions of it can be reproduced from preceding context, and high complexity when it repeatedly introduces new token sequences.

Consider a finite token vocabulary $\mathcal{V}$ and a string $S = s_1 s_2 \cdots s_n \in \mathcal{V}^n$ of length n. For $i \le j$, we write $S_{i:j} = s_i s_{i+1} \cdots s_j$, and denote the prefix ending at j by $S_{1:j}$. LZ76 parses S into a sequence of substrings, called *components*, using the notions of reproduction, production, and history.

Reproduction. A prefix $S_{1:k}$ is said to be *reproducible* from a shorter prefix $S_{1:j}$, with $j < k$, if the appended substring $S_{j+1:k}$ can be copied starting from an earlier position in the string. Formally, we write $S_{1:j} \to S_{1:k}$ if there exists an index $p \le j$ such that $s_{j+r} = s_{p+r-1}, \quad r = 1, \ldots, k-j$. [1]

Production. Production extends reproduction by one additional token. We say that $S_{1:k}$ is *produced* from $S_{1:j}$, written $S_{1:j} \Rightarrow S_{1:k}$, if $S_{1:j} \to S_{1:k-1}$. In other words, all but the final token of the appended substring can be reproduced, while the final token s_k is unconstrained and may introduce new structure. A production appends a component $F = S_{j+1:k}$ such that $S_{j+1:k-1}$ is reproducible from $S_{1:j}$ and the final token s_k is called the *extending token* of the component.

Histories. A *history* of S is a factorization $S = F_1 F_2 \cdots F_m$ such that every component is obtained by a valid production from the prefix formed by the preceding components. Equivalently, let $0 = h_0 < h_1 < \cdots < h_m = n$ denote the component boundaries, with $F_i = S_{h_{i-1}+1:h_i}$. Then the factorization is a history if $S_{1:h_{i-1}} \Rightarrow S_{1:h_i}$ for every i.

Exhaustive History. A component F_i is *exhaustive* if it extends reproduction as far as possible before introducing an extending token. Formally, $S_{1:h_{i-1}} \Rightarrow S_{1:h_i}, \quad S_{1:h_{i-1}} \not\Rightarrow S_{1:h_i}$. That is, the component excluding its final token can be reproduced from the preceding prefix, whereas the com-

[1] The source is required to begin inside the existing prefix but it may overlap the substring being reproduced as the copy progresses. In other words, tokens reproduced earlier in the same operation may themselves be used to reproduce subsequent tokens. This recursive copying is part of the standard LZ76 definition. We introduce a non-overlapping restriction later when connecting LZ complexity to extractive speculation.

plete component cannot. An *exhaustive history* is a history in which every component is exhaustive, except possibly the final component. The exhaustive history is unique and optimal as it contains the minimum number of components among all histories of S (Lempel & Ziv, 1976).

Definition 1 (LZ complexity)**.** *Let $S = F_1F_2 \cdots F_{c(S)}$ be the exhaustive LZ76 history of S. The number of components $c(S)$ is called the* Lempel–Ziv complexity *of S.*

LZ complexity measures the rate at which a string introduces new structure. Strings containing substantial repetition admit fewer and longer components, whereas strings containing little repetition require many short components. We provide a reference Python implementation in Appendix A and illustrate the LZ76 parsing of an example string in Figure 2.

3 Bounding the Acceptance Length of Extractive Speculation

We now connect Lempel–Ziv complexity to extractive speculation. Drafter is extractive if every draft is a substring of the context. In this section, we theoretically answer the question *What is the maximum acceptance length for an extractive speculative drafter?*

Consider a sequence of tokens $S = PG$ which includes the prefill (P) and the generation (G). At a speculation step at position $t > |P|$, the available context is $S_{<t}$. An *extractive draft* is any string D satisfying $D \subseteq S_{<t}$. If D agrees with the target continuation beginning at t (*i.e.* $D = S_{t:t+|D|}$), the target verifies the draft and additionally produces the following token ν. The step therefore advances by $F = D\nu$. Consequently, a sequence of successful extractive speculation steps induces a factorization of the sequence as $S = PF_1F_2 \cdots F_m, \quad F_i = D_i\nu_i$. where $D_i \subseteq PF_1F_2 \cdots F_{i-1}$.

At this stage, the equivalance between Lempel-Ziv compression and extractive speculation should be clear. Every speculation step has the same structure as an LZ production with a substring reproduced from preceding context followed by one extending token. This equivalence is further illustrated in Figure 2. In LZ compression, a component consists of reproduced tokens followed by an extending token. Whereas in extractive speculation, the reproduced tokens form the accepted draft, while the extending token is the bonus token. Number of components in the history therefore corresponds to a number of target-model calls. Before we state a formal equivalence, we need to handle the prefill and the recursive structure both are missing in the extractive speculation case.

To account for the initial context P, we define the conditional LZ complexity $c(S \mid P)$ by applying the LZ76 construction to S while treating P as already available history and counting only components covering S. The resulting average component length is $\tau_{\mathrm{LZ}}(S \mid P) \coloneqq {}^{|S|}/_{c(S|P)}$.

There is one more technical difficulty which must be resolved before applying Lempel-Zev analysis to extractive speculation. Standard LZ76 reproduction is recursive as the source of a copied substring must begin in the preceding prefix, but it may overlap the substring currently being reproduced. Tokens reproduced earlier in the same production can therefore be used to reproduce later tokens which is more than a verbatim copy. A draft must be selected before verification and must already occur in its entirety in the available context. For example, suppose the available context ends in a single `a` and the continuation contains a long run `aaaa`.... LZ76 may recursively reproduce the run by repeatedly referring to the newly reproduced symbols. An extractive drafter, by contrast, cannot immediately draft the entire run unless that run already occurs in the context.

We therefore only consider a non-recursive restriction of LZ76. This version has been analyzed by Kosolobov & Shur (2019). We defer the analysis in the Appendix B and simply refer $c_{\mathrm{nov}}(S \mid P)$ as the no-overlap version of the conditional Lempel-Ziv complexity. Since no-overlap is a restriction, $c(S \mid P) \leq c_{\mathrm{nov}}(S \mid P)$. In other words, the ordinary LZ76 complexity provides an optimistic bound on extractive speculation, while c_{nov} characterizes it exactly. We now state the central result.

Theorem 1 (Extractive Speculation Bound)**.** *For a fixed prefill P and continuation S, the minimum number of target-model calls required by any extractive speculator is*

$$N_{\mathrm{ext}}(S \mid P) \geq c_{\mathrm{nov}}(S \mid P).$$

Consequently, the maximum average number of accepted tokens is

$$\mathrm{AvgAccLength}(S|P)_{ext} \leq \frac{|S|}{c_{\mathrm{nov}}(S \mid P)}.$$

Table 1: **LZ bounds vs. MTP on SpeedBench using Qwen3.6-27B,** $K{=}8$**. LZ Bound**: prompt only; **+hist**: plus earlier outputs; **+corpus**: plus English Wikipedia (6.4M articles, 4.8B tokens).

	Coding	Human.	Math	Multi.	QA	RAG	Reason.	Roleplay	STEM	Summ.	Writing	Avg.
LZ Bound	3.23	2.45	2.94	2.93	1.73	2.90	2.31	1.71	2.59	2.62	2.20	2.51
+ hist	4.44	3.04	3.90	3.39	2.42	3.78	3.27	2.39	3.28	2.92	2.76	3.24
+ hist + corpus	5.43	4.61	4.86	4.20	4.78	5.26	4.70	4.33	4.57	4.88	4.32	4.72
MTP	5.74	4.11	4.77	4.97	3.69	5.55	4.63	2.71	4.43	4.18	3.35	4.30

Proof. Every extractive speculative decoder with m target-model calls induces an extractive non-overlapping history with m components as the accepted draft is the reproduced part of each component and the bonus token is its extending token. Similarly, every non-overlapping history with m components induces a valid oracle speculation schedule with m calls by using the reproduced part of each component as its draft.

It therefore remains only to minimize the number of components. The dictionary of substrings available at each boundary is suffix-closed, so greedily choosing the longest available match minimizes the number of components among all valid parses (Crochemore et al., 2012). Hence the exhaustive non-overlapping history has the minimum possible number of components, namely $c_{\mathrm{nov}}(S \mid P)$. Second statement is simply averaging as $\mathrm{AvgAccLength}(S|P)_{ext} = {}^{|S|}/_{N^{\star}_{\mathrm{ext}}(S|P)}$. □

3.1 Computing the Bound on Existing Speculative Decoding Benchmarks

A significant implication of our theoretical result is that Lempel–Ziv complexity, specifically its non-recursive variant, can be computed exactly and efficiently for any corpus. Hence, we can now empirically answer the question: *What is the maximum acceptance length achievable by an extractive speculative drafter on a specific benchmark?*

We precisely compute the Lempel–Ziv complexity $c_{\mathrm{nov}}(S \mid P)$ for trajectories from Qwen3.5-9B (Yang et al., 2025) on SWE-Bench (Jimenez et al., 2024) and from Qwen3.5-27B (Yang et al., 2025) on SpeedBench (Abramovich et al., 2026). Concretely, given a request with prompt P and output S, we run the LZ algorithm on the concatenated sequence PS and compute the non-overlapping LZ complexity $c_{\mathrm{nov}}(S \mid P)$, which determines the upper bound on the average acceptance length. In addition, we compare the resulting bound against existing extractive drafters, i.e. Ngram (Saxena, 2023) and Suffix (Oliaro et al., 2025), as well as generative drafter MultiTokenPrediction(MTP) (Qwen Team, 2026) shared as part of the original model.

The acceptance lengths of extractive drafters are bounded by Theorem 1. A natural baseline for this comparison is the generative speculative decoding module, multi-token prediction (MTP), developed as part of Qwen3.6-27B. The empirically computed LZ bound remains below the generative speculator (MTP) on both SpeedBench (Table 1) and SWE-bench (Figure 1). Hence, under the same evaluation setup and copy corpus, no extractive speculator can reach the acceptance length of MTP on these benchmarks. This provides a conclusive negative result for using extractive speculative decoding as a direct replacement for generative speculative decoding under the current benchmark setup, unless the available copy corpus or evaluation setting is changed.

3.2 Larger Copy Corpora Narrow the Extractive–Generative Gap

The empirical results in the previous section demonstrate a substantial gap between extractive and generative speculation under the standard benchmarking setup. We now modify this setup to find when does extractive speculation becomes effective.

Before turning to the empirical study, we first consider the effect of context size theoretically. The Lempel–Ziv complexity is monotonic with respect to the available context. Let P and P' be two contexts such that every substring available for copying under P is also available under P'. Then, $c_{\mathrm{nov}}(S \mid P') \leq c_{\mathrm{nov}}(S \mid P)$, since every extractive parsing valid under P remains valid under P'. Equivalently, every schedule feasible under P is also feasible under P', and minimizing the number of phrases over this enlarged feasible set can only decrease the minima. Consequently, the corresponding upper bound on acceptance length can only increase as the context grows.

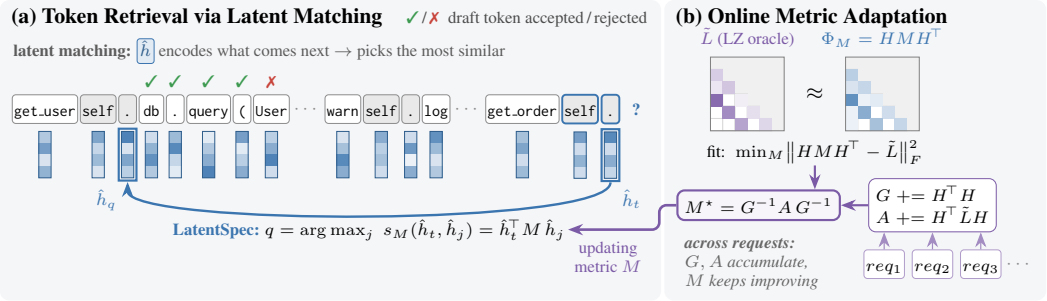


Figure 3: **LatentSpec. (a)** Every token in the context is cached with its normalized last-layer hidden state $\hat{h}_j$ (blue columns). LatentSpec compares the current token $\hat{h}_t$ against the cache using the metric $s_M(\hat{h}_t, \hat{h}_j) = \hat{h}_t^\top M \hat{h}_j$, retrieves the best match $\hat{h}_q$ and copies its continuation $x_{q+1:q+k}$ as the draft. **(b)** The LZ parsing provides supervision for every pair of the served tokens $\tilde{L}$, and we fit the metric M so that the score matrix $\Phi_M = HMH^\top$ approximates $\tilde{L}$. The adaptation is done in closed-form.

This monotonicity provides theoretical justification for enlarging the copy corpus beyond the current request. Existing methods already exploit this to some extent. Suffix Decoding (Oliaro et al., 2025), for example, accumulates reusable text across requests, while REST (He et al., 2024) retrieves candidate continuations from an external datastore. Here, we systematically study how enlarging the copy corpus affects the upper bound on extractive speculation.

We first analyze the effect of context size within SWE-Bench by bucketing requests according to context length, from 1k to 64k tokens. We observe that the LZ bound generally increases with the context length, while the acceptance length of the generative speculator varies over a much smaller range. For example, at a context length of 1k tokens, the gap between MTP and the LZ bound is 2.25 tokens (5.10 vs. 2.85), whereas at 64k tokens the gap decreases to 0.68 tokens (6.45 vs. 5.77).

Extractive speculation, however, need not be restricted to the context of the current request. We therefore perform a second experiment in which we explicitly enlarge the available copy corpus using additional history and repository content from SWE-bench. The *history* consists of sessions solving other issues, while the *codebase* consists of the source code of the corresponding repository. We denote the resulting bounds as *LZ + History* and *LZ + History + Codebase*. We compute these bounds and compare them with the original LZ bound and generative speculation methods in Figure 1. As shown in the figure, enlarging the copy corpus substantially increases the extractive upper bound. With sufficient additional corpus, the LZ bound becomes competitive with generative speculation and can exceed it for longer contexts. A similar trend exists for SpeedBench by using history and an external corpus of English Wikipedia in Table 1. Therefore, the choice and size of the copy corpus are critical when evaluating extractive speculative decoding.

4 Approaching the LZ Bound via Retrieval in Latent Space

In addition to establishing a performance bound for extractive speculation, we use LZ-bound to evaluate existing extractive speculators and precisely characterize their remaining room for improvement. We evaluate *Ngram*, which drafts by exact n-gram lookup over the context (Saxena, 2023), and *Suffix Decoding* (Oliaro et al., 2025), which strengthens n-gram lookup with a suffix tree built over both the current request and previously generated outputs, as well as visit-frequency-based scoring. Results are shown in Figure 4 and Table 2. All existing extractive speculators perform well below the LZ bound, suggesting that substantial room for improvement remains within the extractive setting. We improve them using inspirations from LZ-theory.

Extractive speculative decoding is fundamentally a retrieval problem. At every decoding step, the drafter must select a position in the available context and reuse the continuation that followed it. The LZ bound characterizes the best possible match of this retrieval problem and provide a ground truth retrieval as a side product. The difference between the retrieved token of a practical extractive drafter and the LZ bound reflects how well the drafter works.

Ngram and Suffix Decoding perform retrieval through exact token matching. This is attractive because it is simple and inexpensive, but it uses only token identity to determine which previous context is likely to have a useful continuation. The target model itself provides a richer representation. Its hidden state summarizes the preceding context and is trained to predict what follows. We therefore ask whether the target model's representation space can provide a better retrieval signal for extractive speculation. The resulting method remains fully extractive with no significant overhead as the latent representation is already computed and only an extra nearest neighbor search is needed.

Retrieval in the Model's Representation Space Let $h_t \in \mathbb{R}^{d_h}$ denote the target model's hidden state at token t. We maintain a cache of previous hidden states with the continuations that followed them. We retrieve the token with the closest state using a Mahalanobis distance in the normalized space: $\arg\max_{j:j+k\leq t} \hat{h}_t^\top M \hat{h}_j$ where $\hat{h} = {}^{h}/_{\|h\|_2}$, and use the continuation $x_{j+1:j+k}$ as the draft. Both the representations and tokens are by-products of decoding, so maintaining the cache has no overhead beyond memory storage. We visualize this process in Figure 3.

The Similarity Metric: The choice of similarity function is the central component of any retrieval problem. Non-informative metric with $M = I$ collapses to cosine similarity and is already effective; however, we can do much better since LZ analysis provides exactly the supervision needed to adapt this similarity. Given a sequence of tokens with length n, LZ algorithm produces a $L \in \mathbb{N}^{n\times n}$ matrix such that each entry corresponds to prefix match length which is the number of tokens that agree between the continuations at these indices.

We use this signal to adapt the representation-space similarity so that pairs of hidden states with long matching continuations receive higher scores. In particular, we fit the bilinear similarity matrix M to match this prefix match length matrix L by minimizing $\left\|H\, M\, H^\top - L\right\|_F^2$.

This gives a natural online adaptation mechanism. We initially use the model's representation space with $M = I$ and continuously update its similarity function with every token. The same adaptation can also be performed on the current prompt during prefill, since both its hidden states and future tokens within the prompt are already known. The method remains extractive while progressively adapting its retrieval rule toward the optimal decisions identified by the LZ oracle. We derive this online adaptation method in Appendix C.

4.1 Evaluating the Retrieval in the Latent Space

Settings We evaluate on three models: Qwen3.5-{9B,27B} and Qwen3.6-27B. We use SpeedBench (short-context, chat) from Abramovich et al. (2026) as well as SWEBench (long-context, agentic) from Jimenez et al. (2024) (verified-split). We primarily report *acceptance length* (AL), the average number of tokens accepted per forward pass (including the bonus token); *throughput* (tokens per second), which captures end-to-end serving efficiency, is reported in Appendix E. We use vLLM's (Kwon et al., 2023) native implementations of Ngram and Suffix decoding, and implement LatentSpec in the same vLLM fork. All drafters therefore share the same scheduler, model inference kernels and speculative decoding pipeline. More experiment details can be found in Appendix D.

SWEbench Results Figure 4 reports the acceptance length of the extractive speculators under different context settings. Three trends hold consistently: (i) there is a large gap between the LZ bound and the acceptance length realized by the three extractive speculators. This large gap illustrates the challenge of accurately retrieving the correct token spans from the context. (ii) While adding history requests into the context consistently improves the acceptance length, somewhat counter-intuitively, appending the *codebase* on top of history *lowers* acceptance length. This is because while more context brings more sources to copy from, it also greatly increases the retrieval challenge. (iii) LatentSpec matches or exceeds Suffix in every panel, and strictly outperforms Ngram.

Speedbench Results Table 2 reports the same comparison on SpeedBench (Qwen3.5-9B), broken down by category, and the trends we observe are largely consistent with Figure 4. The gap between the LZ bound and the realized acceptance length remains large across every category, and widens further once history is added to the context (e.g. on Coding, the bound grows from 2.79 to 3.85 while the best practical method only moves from 1.65 to 1.80), reaffirming that retrieving the right span becomes harder, not easier, as more context is exposed. Unlike on SWEbench (coding only),

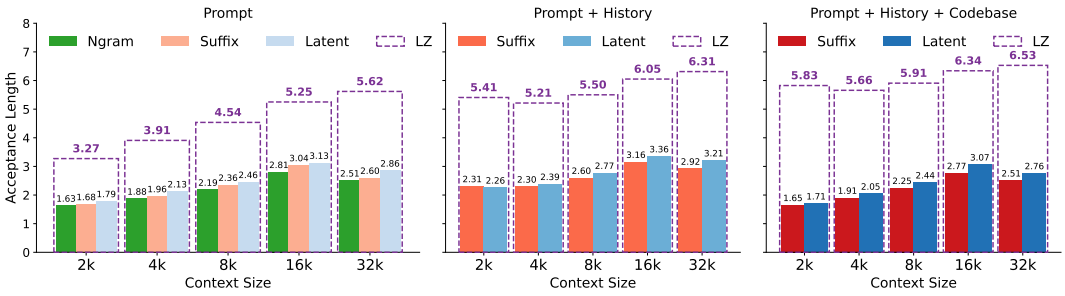


Figure 4: **Acceptance length of extractive speculators on SWEbench (Qwen3.5-9B, K=8).** The three panels correspond to the three context settings: the running request alone, plus *history* (sessions solving other issues), and plus the *codebase* (source of the corresponding repository). Within each panel, we plot the measured acceptance length of the three candidate methods under growing contexts. The dashed bracket in each panel marks the LZ oracle upper bound. Ngram is omitted from the right two panels: it cannot accumulate context beyond its current window, and its matching is strictly weaker than Suffix.

Table 2: **Acceptance length of extractive speculators on SpeedBench (Qwen3.5-9B, K=8), by category.** Ngram/Suffix/Latent are extractive speculators; LZ Bound is the upper bound for the corresponding configuration. Columns are grouped by the available information: the prompt alone, the prompt plus history, and the prompt plus history and corpus.

	Prompt				+History			+History and Corpus		
Category	Ngram	Suffix	Latent	LZ Bound	Suffix	Latent	LZ Bound	Suffix	Latent	LZ Bound
Coding	1.52	1.55	**1.65**	2.79	1.70	**1.80**	3.85	1.51	**1.66**	4.85
Math	1.52	1.56	**1.72**	2.88	1.69	**1.85**	3.86	1.61	**1.75**	4.80
Multilingual	1.60	1.62	**1.70**	3.10	1.72	**1.79**	3.61	1.69	**1.87**	4.38
RAG	1.57	**1.59**	1.55	2.74	**1.70**	1.66	3.49	**1.73**	1.72	5.10
Summarization	1.41	**1.42**	1.36	2.52	**1.47**	1.40	2.84	**1.44**	1.43	4.71
STEM	1.49	1.53	**1.59**	2.52	1.62	**1.67**	3.23	1.64	**1.68**	4.56
Humanities	1.49	**1.53**	1.49	2.46	1.57	**1.62**	3.07	1.54	**1.56**	4.82
Reasoning	1.34	1.41	**1.43**	2.23	1.55	**1.58**	3.10	**1.54**	1.53	4.57
Writing	1.22	1.25	**1.26**	2.14	**1.32**	1.29	2.70	1.27	**1.29**	4.32
QA	1.28	1.29	**1.45**	1.74	1.42	**1.44**	2.46	1.67	**1.72**	4.98
Roleplay	1.07	**1.09**	1.08	1.71	**1.16**	1.12	2.44	**1.20**	1.12	4.42
Average	1.42	1.45	**1.50**	2.48	1.55	**1.60**	3.21	1.53	**1.57**	4.63

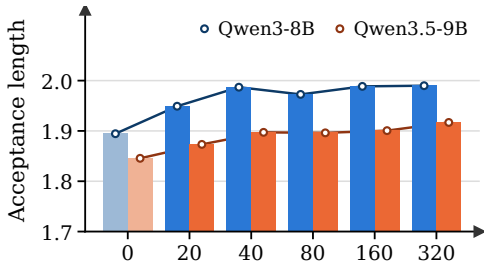
no single method dominates across all categories: LatentSpec is the better on Coding, Math, while Suffix is consistently better on RAG, Summarization.

Online Adaptation Ablation We ablate the online adaptation component on ShareGPT, a large collection of real-world user–chatbot conversations. We let the drafter observe the first N requests to accumulate its statistics, and then measure acceptance length on the next 300 requests. Table 3(left) shows a consistent improvement in acceptance length across all five models. Right panel plots acceptance length as a function of N: the gain appears almost immediately, with 20 requests already recovering more than half of it, and saturates by roughly 40.

The Large Gap to the LZ Bound. Across both benchmarks, every extractive drafter we evaluate, including ours, realizes only a fraction of the acceptance length permitted by the LZ bound. On SpeedBench, LatentSpec attains 1.50 of the 2.48 available from the prompt alone (60%), and 1.60 of 3.21 once history is added (50%). We consider two reasons. First, the LZ bound is oracle as it identifies every exact match, including matches that may be difficult to infer from semantic similarity. Second, retrieval is challenging and becomes an increasingly important bottleneck as the search space expands, as indicated by the widening gap with increasing context size. LatentSpec narrows this gap but still remains substantially below the LZ bound and non-extractive speculators such as native MTP.

Table 3: **Ablation of online adaptation.** (left) Acceptance length with and without online adaptation on ShareGPT. (right) Acceptance length as a function of the number of requests processed, showing how the drafter improves as it observes more traffic.

	Acceptance length	
Model	w/o Adapt.	with Adapt.
Qwen3-8B	1.89	**1.99**
Qwen3.5-9B	1.85	**1.92**
Qwen3.5-35B	1.57	**1.59**
Qwen3.6-27B	1.41	**1.44**
Qwen3.6-35B	1.48	**1.49**



5 Related Works

Theoretical Analysis of Speculative Decoding. Yin et al. (2024) analyse speculative decoding with a model-based drafter, showing that the acceptance rate α is governed by the total variation distance between the draft and target distributions. Sun et al. (2024) likewise bound the acceptance rate for a given draft distribution, casting verification as an optimal-transport problem.

Compression and Language Model. Delétang et al. (2024) show that a pretrained language model paired with arithmetic coding is a state-of-the-art general-purpose compressor, making the prediction-compression equivalence concrete. Our work draws another connection between language modelling and compression, this time from the speculative decoding perspective and with a dictionary-based compression algorithm. Zip2zip (Geng et al., 2025) applies LZW parsing at inference time to merge recurring token sequences into adaptive vocabulary units. We use a similar parsing algorithm, but as an analysis tool rather than a modelling device.

Speculative Decoding using Hidden States Pal et al. (2023) show that the hidden states of pretrained language models encode information not only about the immediate next token, but also about subsequent tokens, and that this information can be extracted via linear probes. Building on this insight, recent work on multi-token prediction (Gloeckle et al., 2024; DeepSeek-AI, 2025) trains auxiliary modules to decode future tokens $x_{t+1:t+k}$ from the hidden state h_t at position t. Speculative decoding methods such as Medusa (Cai et al., 2024) and EAGLE (Li et al., 2024b) exploit the same property, training lightweight prediction heads on top of the target model's hidden states to produce draft tokens. LatentSpec instead uses the hidden states to *retrieve* a matching span from the past rather than to decode tokens.

6 Conclusion

We studied extractive speculative decoding, where drafts are copied from previously observed text rather than generated by an auxiliary model. We established a precise connection between extractive speculation and Lempel–Ziv compression and derived a tight and computable *LZ bound* on the acceptance length achievable by any extractive drafter. As a negative result, we provably found that the context fails to provide sufficient reusable text for extractive methods to match generative speculators. However, enlarging the copy corpus with interaction history and repository code, substantially raises this ceiling and can make extractive speculation competitive. Even with a sufficiently rich corpus, existing extractive methods stay well below the theoretical limit. To address this, we introduced LatentSpec, which retrieves candidate spans in the target model's representation space and adapts its similarity metric online using LZ-derived supervision. LatentSpec improves over existing extractive drafters, but still remains well below the LZ bound. The retrieval remains as the central open problem for extractive speculation and we believe that the LZ bound can serve as a useful guide for designing future retrieval methods, copy corpora, and evaluation settings.

AI USE STATEMENT

We used generative AI tools—both agentic assistants and inline code completion—at several stages of this work.

Among the tasks for which ICLR requires disclosure, we used generative AI to help develop the conceptual framing of extractive speculative decoding in terms of Lempel–Ziv parsing and to phrase the statement of the resulting bound (though the idea was originated from authors rather than AI); to articulate and sharpen hypotheses about our empirical observations; to obtain feedback on our experimental design, including identifying an inconsistency in our run configurations; to help implement components of our method and its integration into vLLM; to parse and reformat evaluation logs into the tables reported here; and to interpret our measurements. Generation of synthetic datasets is not applicable to this work, as all experiments use existing public benchmarks.

Additionally, we used generative AI for tasks with recommended disclosure: editing the manuscript for readability, drafting and revising individual paragraphs, mock reviewing, formatting LaTeX tables, brainstorming, searching for and summarising related work, and organising the structure of the paper.

We have reviewed all AI-assisted work. AI-generated code was tested by the authors; no experiment was run autonomously by an AI tool, and every number reported in this paper was recomputed from the authors. Cited references were checked to exist and to support the claims we attribute to them. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with AI assistance.

REFERENCES

Talor Abramovich, Maor Ashkenazi, Izzy Putterman, Benjamin Chislett, Tiyasa Mitra, Bita Darvish Rouhani, Ran Zilberstein, and Yonatan Geifman. Speed-bench: A unified and diverse benchmark for speculative decoding, 2026. URL https://arxiv.org/abs/2604.09557.

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 *ICML*, 2024.

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

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. *arXiv preprint arXiv:2602.06036*, 2026.

Jacob K Christopher, Brian R. Bartoldson, Tal Ben-Nun, Michael Cardei, Bhavya Kailkhura, and Ferdinando Fioretto. Speculative diffusion decoding: Accelerating language generation through diffusion. In *NAACL*, 2025.

Maxime Crochemore, Alessio Langiu, and Filippo Mignosi. Note on the greedy parsing optimality for dictionary-based text compression, 2012. URL https://arxiv.org/abs/1211.5350.

DeepSeek-AI. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. In *ICLR*, 2024.

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. CopySpec: Accelerating LLMs with speculative copy-and-paste. In *EMNLP*, 2025.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed 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 (eds.), *ACL*, 2024.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding, 2024. URL https://arxiv.org/abs/2402.02057.

Saibo Geng, Nathan Ranchin, Yunzhen yao, Maxime Peyrard, Chris Wendler, Michael Gastpar, and Robert West. zip2zip: Inference-time adaptive vocabularies for language models via token compression, 2025. URL https://arxiv.org/abs/2506.01084.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *ICML*, 2024.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. REST: Retrieval-based speculative decoding. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *NAACL*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

Dmitry Kosolobov and Arseny M. Shur. Comparison of lz77-type parsings. *Information Processing Letters*, 141:25–29, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, 2023.

Alan Chi-Man Lee, Wing-Sun Cheng, and Calvin Chun-Kit Chan. PROMTEC: Fast LLM inference decoding using prompt multi-lookup with template database and common sequences. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *ACL*, 2025.

A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *EMNLP*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: speculative sampling requires rethinking feature uncertainty. In *ICML*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *NeurIPS*, 2025.

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 *NeurIPS*, 2024.

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

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 *ASPLOS*, 2024.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. Suffixdecoding: Extreme speculative decoding for emerging ai applications, 2025. URL https://arxiv.org/abs/2411.04975.

Koyena Pal, Jiuding Sun, Andrew Yuan, Byron Wallace, and David Bau. Future lens: Anticipating subsequent tokens from a single hidden state. In Jing Jiang, David Reitter, and Shumin Deng (eds.), *CoNLL*, 2023.

Guofeng Quan, Wenfeng Feng, Chuzhan Hao, Guochao Jiang, Yuewei Zhang, and Hao Henry Wang. RASD: Retrieval-augmented speculative decoding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *ACL*, 2025.

Qwen Team. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. Technical report, Alibaba Group, August 2026.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential, 2025. URL https://arxiv.org/abs/2507.11851.

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

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport, 2024. URL https://arxiv.org/abs/2310.15141.

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, 2025. URL https://arxiv.org/abs/2505.09388.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. Inference with reference: Lossless acceleration of large language models, 2023. URL https://arxiv.org/abs/2304.04487.

Ming Yin, Minshuo Chen, Kaixuan Huang, and Mengdi Wang. A theoretical perspective for speculative decoding algorithm. In *NeurIPS*, 2024.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft and verify: Lossless large language model acceleration via self-speculative decoding. In *ACL*, 2024.

A Reference Implementation of Non-Overlapping LZ76

The following Python function computes the LZ76 factorization of an integer sequence (e.g. token IDs). It returns the complexity $c(S)$ and the list of phrases. The algorithm uses an explicit double loop for clarity: the outer loop emits one phrase per iteration; the inner loop scans the history window $[0, l)$ for the longest match.

B Non-Overlapping Conditional Lempel–Ziv Complexity

Although the recursive behavior in LZ76 is useful for compression, it is not available to an extractive speculative drafter as the complete draft must already exist in the context when the target-model call begins. We therefore use a non-overlapping variant of LZ complexity studied in prior work (Kosolobov & Shur, 2019). We also directly study the conditional version using the prefill P. We first define the non-overlapping versions of the reproduction, production and extensive histories.

Consider an initial prefill P that is already available as context and the string $S = s_1 s_2 \cdots s_n$. At position i in S, the available history is $PS_{1:i-1}$.

Non-overlapping reproduction. A prefix $S_{1:k}$ is said to be *non-overlapping reproducible* from a shorter prefix $S_{1:j}$, with $j < k$, if the appended substring $S_{j+1:k}$ occurs entirely within the preceding context. Formally, we write $S_{1:j} \rightarrow_{\text{nov}} S_{1:k}$ if $S_{j+1:k} \subseteq PS_{1:j}$. Unlike standard LZ76 reproduction, the source must lie completely within the context available before reproduction begins.

```python
def lz76(ss):
    """LZ76 factorization of an integer sequence ss.
    Returns (c, phrases): c = complexity, phrases = list of lists."""
    ss = list(ss)
    n = len(ss)
    if n == 0:
        return 0, []

    phrases = [ss[0:1]]
    c = 1
    l = 1  # start of current phrase

    while l < n:
        k_max = 1  # at least 1 (single new token)
        for i in range(l):  # scan history [0, l)
            k = 1
            while (l + k <= n
                   and ss[i + k - 1] == ss[l + k - 1]):
                k += 1
            if k > k_max:
                k_max = k

        phrase_end = min(l + k_max, n)
        phrases.append(ss[l:phrase_end])
        c += 1
        l = phrase_end

    return c, phrases
```

Figure 5: Reference Python implementation of LZ76 factorization. Complexity $O(n^2)$ in the worst case. Each phrase = longest match in history + one new token (the final phrase may omit the new token if the sequence ends mid-match).

Non-overlapping production. Non-overlapping production extends non-overlapping reproduction by one additional token. We say that $S_{1:k}$ is *non-overlapping produced* from $S_{1:j}$, written $S_{1:j} \Rightarrow_{\text{nov}} S_{1:k}$, if $S_{1:j} \rightarrow_{\text{nov}} S_{1:k-1}$. Thus, a production appends a component $F = S_{j+1:k}$ such that $S_{j+1:k-1}$ already occurs entirely in the preceding context, while the final token s_k is unconstrained and serves as the extending token.

Non-overlapping histories. A *non-overlapping history* of S conditioned on P is a factorization $S = F_1 F_2 \cdots F_m$ such that every component is obtained by a valid non-overlapping production from P and the preceding components. Equivalently, let $0 = h_0 < h_1 < \cdots < h_m = n$ denote the component boundaries, with $F_i = S_{h_{i-1}+1:h_i}$. Then the factorization is a non-overlapping history if $S_{1:h_{i-1}} \Rightarrow_{\text{nov}} S_{1:h_i}$ for every i, with P treated as pre-existing context.

Exhaustive non-overlapping history. A component F_i is *exhaustive* if it extends non-overlapping reproduction as far as possible before introducing an extending token. Formally, $S_{1:h_{i-1}} \Rightarrow_{\text{nov}} S_{1:h_i}$, $S_{1:h_{i-1}} \not\Rightarrow_{\text{nov}} S_{1:h_i}$. That is, the component excluding its final token can be copied entirely from the preceding context, whereas the complete component cannot. An *exhaustive non-overlapping history* is a non-overlapping history in which every component is exhaustive, except possibly the final component.

As in the standard LZ construction, greedily selecting the longest valid component minimizes the number of components among all non-overlapping histories (Crochemore et al., 2012).

Definition 2 (Non-overlapping LZ complexity). *Let $S = F_1 F_2 \cdots F_{c_{\text{nov}}(S|P)}$ be the exhaustive non-overlapping history of S conditioned on P. The number of components $c_{\text{nov}}(S \mid P)$ is called the* conditional non-overlapping Lempel–Ziv complexity.

The only difference from standard LZ76 is the restriction on reproduction. Standard LZ76 permits recursive copying, whereas the non-overlapping variant requires the entire reproduced substring to already exist in the preceding context. Since non-overlapping reproduction is strictly more restrictive,

$$c(S \mid P) \leq c_{\mathrm{nov}}(S \mid P).$$

This restriction makes the correspondence with extractive speculation exact. At a speculation step, the accepted draft must already occur in the available context, and the target-model call then produces one additional bonus token. Each successful extractive speculation step therefore corresponds exactly to one non-overlapping LZ production, with the accepted draft as the reproduced substring and the bonus token as the extending token.

Conversely, every non-overlapping production defines a valid oracle extractive speculation step by using its reproduced substring as the draft. Hence, an extractive speculation schedule with m target-model calls is equivalent to a non-overlapping history with m components. Since the exhaustive non-overlapping history minimizes the number of components, it follows that

$$N_{\mathrm{ext}}(S \mid P) \geq c_{\mathrm{nov}}(S \mid P),$$

where $N_{\mathrm{ext}}(S \mid P)$ denotes the number of target-model calls required by any extractive speculator. Consequently,

$$\mathrm{AvgAccLength}_{\mathrm{ext}}(S \mid P) \leq \frac{|S|}{c_{\mathrm{nov}}(S \mid P)}.$$

Thus, ordinary LZ76 complexity provides an optimistic bound because it permits recursive reproduction, whereas the non-overlapping complexity c_{nov} exactly captures the restriction imposed by extractive speculation.

C Online Adaptation

For any sequence S, LZ algorithm defines a *label matrix* $L \in \mathbb{N}^{n \times n}$, where $L_{i,j}$ $(j < i)$ records the number of consecutive tokens that agree between the continuations at positions i and j:

$$L_{i,j} \;=\; \max\{k \geq 0 : s_{i+1:i+k} = s_{j+1:j+k}\}.$$

A perfect retriever would, for every query position i, rank $\arg\max_j L_{i,j}$ first. The goal of metric learning is therefore to find M such that the similarity ranking approximates the label ranking as closely as possible. In principle, speculative decoding is a sequential decision process: the position decoded at step $k{+}1$ depends on how many tokens are accepted at step k, so later states are consequences of earlier retrieval choices. Solving this exactly would require reinforcement learning, but for efficiency we make a simplifying approximation: we treat each position independently and learn a metric that predicts the best match at every position in isolation, ignoring the sequential dependence.

Objective. Let $H \in \mathbb{R}^{n \times d_h}$ stack the normalised hidden states $\hat{h}_i$ row-wise, and let $\tilde{L}$ denote the label matrix with the diagonal and upper triangle masked out. The score matrix over all position pairs is then $\Phi_M = H\,M\,H^\top$, with $(\Phi_M)_{i,j} = s_M(h_i, h_j)$. We seek the M that minimises the Frobenius reconstruction error between this score matrix and the label matrix:

$$\mathcal{L}(M) \;=\; \left\| H\,M\,H^\top - \tilde{L} \right\|_F^2.$$

Closed-form solution. Because the mapping $M \mapsto HMH^\top$ is linear in M, this objective is a convex quadratic and admits a closed-form solution. Setting the matrix gradient to zero,

$$\frac{\partial\,\mathcal{L}}{\partial M} \;=\; 2\,H^\top\big(HMH^\top - \tilde{L}\big)\,H \;=\; 0,$$

yields the normal equation

$$(H^\top H)\,M\,(H^\top H) \;=\; H^\top \tilde{L}\,H.$$

When $H^\top H$ is rank-deficient (as is typical when $n < d_h$), we regularise by replacing $H^\top H$ with $G = H^\top H + \lambda I$, giving the closed-form solution

$$M^* = G^{-1}\,H^\top \tilde{L}\,H\,G^{-1}, \qquad G = H^\top H + \lambda I.$$

Note that the normal equation factors into two $d_h \times d_h$ systems rather than a single $d_h^2 \times d_h^2$ one: this is a consequence of the Kronecker structure of the problem, and keeps the total cost at $O(n^2 d_h + d_h^3)$.

Online update via sufficient statistics. The solution $M^* = G^{-1} A G^{-1}$ depends on only two $d \times d$ matrices: $G = H^\top H + \lambda I$ and $A = H^\top \tilde{L}\, H$. Both are additive in data: when a new sample $(H_{\text{new}}, L_{\text{new}})$ arrives, update $G \leftarrow G + H_{\text{new}}^\top H_{\text{new}}$ and $A \leftarrow A + H_{\text{new}}^\top \tilde{L}_{\text{new}}\, H_{\text{new}}$, then recompute $M^* = G^{-1} A G^{-1}$. This enables *continuous adaptation*: the metric improves as more requests are served, without storing raw hidden states or re-solving from scratch. In the online-at-prefill setting, M is fit from the current prompt's hidden states alone — providing per-request adaptation at zero extra data-collection cost.

D Additional Experimental Details

Table 4 reports the mean prompt and generation lengths for each SpeedBench category. Both are short: prompts average 323 tokens and do not exceed 900 even in the longest category (RAG), while generations average 726 tokens and stay below 1600 throughout. This makes SpeedBench an unfavourable setting for extractive speculation. With so little text in the window there is simply not much material to copy from. By contrast, the long-context, history-rich agentic workloads of SWEbench provide the complementary regime, one in which the context accumulates enough redundancy for extractive speculation to be effective.

Table 4: **SpeedBench request statistics per category (Qwen3.6-27B).** *In*/*Out*: mean prompt and generated tokens per sample; *n*: number of samples.

	Coding	**Human.**	**Math**	**Multi.**	**QA**	**RAG**	**Reason.**	**Roleplay**	**STEM**	**Summ.**	**Writing**	**Avg.**
n	80	80	80	80	80	80	80	80	80	80	80	880
In	245	111	137	170	23	882	114	471	129	612	662	323
Out	490	1035	1486	419	340	209	632	267	1370	212	1530	726

SWEbench Results on Two More Models We replicate the same experiments on SWEbench with two more models: Qwen3.5-27B and Qwen3.5-27B. We report the SWEbench results in Figure 6. We omit the third context configuration (plus *codebase*), as it requires contexts of up to 256K tokens and is thus computationally prohibitive to run at this model scale; we expect its behaviour to closely mirror the Qwen3.5-9B result in Figure 4.

Speedbench Results on Two More Models We replicate the same experiments on speedbench with two more models: Qwen3.5-27B and Qwen3.5-27B. We report the results in Table 5. We omit the third context configuration (plus *codebase*), as it requires contexts of up to 256K tokens and is thus computationally prohibitive to run at this model scale; we expect its behaviour to closely mirror the Qwen3.5-9B result in Table 2.

E Wall-Clock Efficiency

Speculative decoding is ultimately a technique for accelerating inference, so what matters in deployment is end-to-end throughput, measured in tokens per second. The main focus of this work is acceptance length, which isolates the quality of the drafter from the engineering of the serving stack; but acceptance length translates into speedup only up to the cost of producing and verifying the drafts. Concretely, for any speculative method the speedup over autoregressive decoding is

$$\text{speedup} \;=\; \frac{\text{AL}}{1 + c_{\text{draft}} + c_{\text{verify}}} \;\le\; \text{AL} \;\le\; \text{LZ}, \tag{1}$$

where c_{draft} is the overhead of constructing a draft and c_{verify} the overhead of verifying k candidate tokens in a single forward pass, both expressed relative to one target model forward step. Equation 1 makes the ceiling explicit. Even a drafter with zero overhead cannot exceed an AL$\times$ speedup, and AL itself is capped by the LZ bound of Section 3.

As Figure 4 shows, existing extractive drafters remain far from the LZ bound and further still from generative drafters such as MTP. The main bottleneck is thus acceptance length.

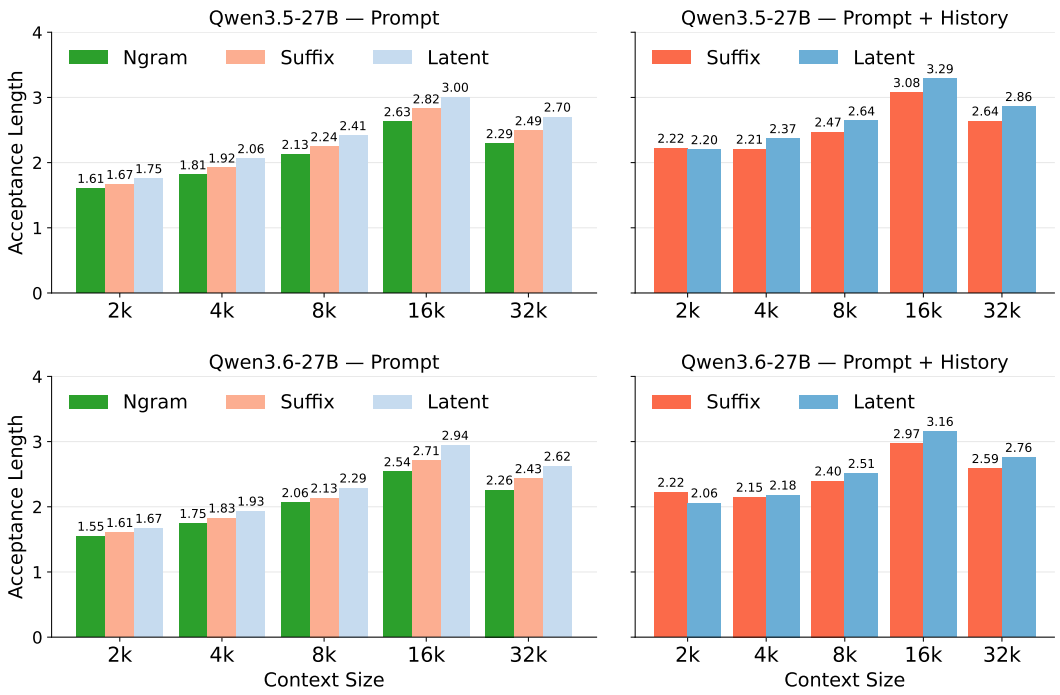


Figure 6: **Acceptance length of extractive speculators on SWEbench (Qwen3.5-27B and Qwen3.5-27B, K=8)**. Same setting as in Figure 4.

Efficient Implementation. To implement LatentSpec efficiently and support production-scale inference engine such as vLLM (Kwon et al., 2023), we have made the following design choices: *(1) Sequence packing.* We pack the hidden states of multiple concurrent requests into a single batch, and perform a single matrix multiplication to compute the similarity scores against all cached keys. This allows us to leverage GPU parallelism and reduce the overhead of multiple small matrix multiplications. *(2) Sync-free GPU drafting.*The entire retrieval path stays on the GPU — scoring, ranking and draft assembly are all device operations, and the drafts are handed back to the engine as a device tensor rather than as a host-side list. Retrieval therefore introduces no host–device synchronisation on of its own, and the decoding loop never stalls waiting for a draft to make the round trip to the CPU.

We measure the throughput of the three extractive drafters on both SpeedBench (short-context, chat) and SWE-Bench (long-context, agentic) using the built-in vLLM profiler, with vLLM pinned at version 0.17.0. Figure 7 and Figure 8 report the results on the two benchmarks respectively. The advantage on acceptance length translates to throughput, but the margins shrink considerably. Drafting cost grows in the order (Ngram $<$ Suffix $<$ LatentSpec), so the three methods lie within 2% of one another in end-to-end throughput.

One point deserves attention. On SWE-Bench, online adaptation introduces a noticeable overhead and *lowers* end-to-end throughput (Figure 8): despite achieving a higher acceptance length than LatentSpec alone, it is slower in wall-clock terms. The cause is the cost of applying a metric update. The online adaptation periodically computes the new metric M^* (Appendix C) after a certain number of requests accumulated.

When an update is triggered, we have the old metric and the new metric and this creates a consistency challenge: requests that are already partially generated hold latent states projected under the old metric, and once the new metric takes effect these states must be reprojected for the entire context accumulated so far. On SWE-Bench, where contexts reach the order of 100K tokens, this reprojection is expensive enough to outweigh the gain in acceptance length. This could likely be mitigated by reducing the update frequency, or by stopping updates altogether once the improvement in acceptance length saturates, as shown in Figure 3(right). We leave this to future work.

Table 5: **Acceptance length of extractive speculators on SpeedBench (Qwen3.5-27B and Qwen3.5-27B, K=8)**. Same setting as in Table 2.

	Prompt					+History			
	Ngram	Suffix	Latent	Latent-OL	LZ	Suffix	Latent	Latent-OL	LZ
Qwen3.5-27B									
Coding	1.46	1.50	1.60	**1.62**	2.64	1.64	1.74	**1.79**	3.66
Humanities	1.43	1.51	**1.56**	1.52	2.34	1.55	**1.61**	1.58	2.93
Math	1.59	1.62	1.71	**1.75**	2.92	1.75	1.84	**1.92**	3.91
Multilingual	1.55	1.58	1.66	**1.67**	2.86	1.66	1.76	**1.78**	3.31
QA	1.33	1.44	**1.50**	1.30	1.89	1.32	1.45	**1.47**	2.61
RAG	1.56	**1.58**	1.51	1.55	2.72	**1.68**	1.64	1.68	3.48
Reasoning	1.35	1.39	1.45	**1.46**	2.23	1.57	1.60	**1.61**	3.15
Roleplay	1.07	1.09	1.09	**1.09**	1.72	**1.15**	1.12	1.12	2.41
STEM	1.49	1.56	1.60	**1.62**	2.63	1.64	1.69	**1.70**	3.34
Summarization	1.40	**1.40**	1.34	1.35	2.49	**1.45**	1.38	1.39	2.83
Writing	1.21	1.25	**1.26**	1.26	2.15	**1.32**	1.32	1.31	2.74
Average	1.40	1.46	1.51	**1.51**	2.47	1.52	1.56	**1.58**	3.21
Qwen3.6-27B									
Coding	1.65	1.70	1.77	**1.82**	3.23	1.89	1.90	**1.98**	4.44
Humanities	1.43	1.54	1.54	**1.55**	2.45	1.56	1.57	**1.61**	3.04
Math	1.53	1.58	1.70	**1.74**	2.94	1.68	1.84	**1.88**	3.90
Multilingual	1.63	1.65	**1.75**	1.75	2.93	1.75	1.85	**1.88**	3.39
QA	1.29	1.32	1.26	**1.37**	1.73	1.37	1.35	**1.40**	2.42
RAG	1.65	**1.67**	1.59	1.62	2.90	**1.83**	1.70	1.77	3.78
Reasoning	1.38	1.41	1.49	**1.52**	2.31	1.56	1.65	**1.67**	3.27
Roleplay	1.08	1.10	1.11	**1.11**	1.71	**1.16**	1.15	1.14	2.39
STEM	1.47	1.51	1.56	**1.60**	2.59	1.61	1.65	**1.67**	3.28
Summarization	1.43	**1.43**	1.36	1.38	2.62	**1.49**	1.41	1.42	2.92
Writing	1.23	1.27	**1.29**	1.28	2.20	1.33	**1.34**	1.33	2.76
Average	1.43	1.47	1.52	**1.54**	2.54	1.57	1.58	**1.61**	3.28

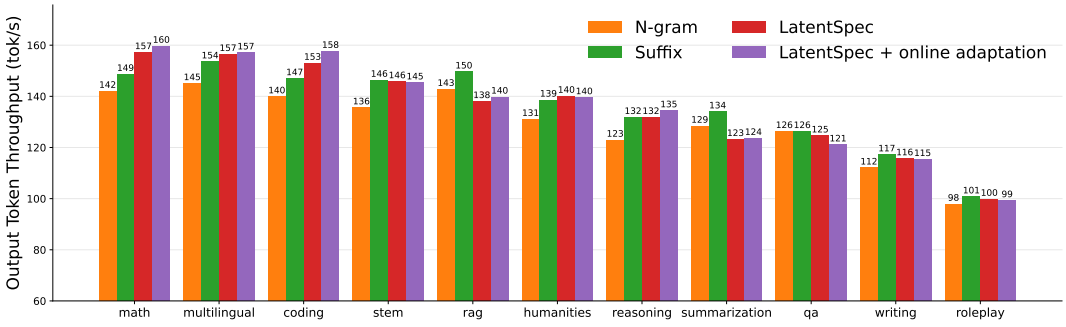


Figure 7: **Throughput measurement on speedbench (Qwen3.5-9B, BS = 1, A100)**

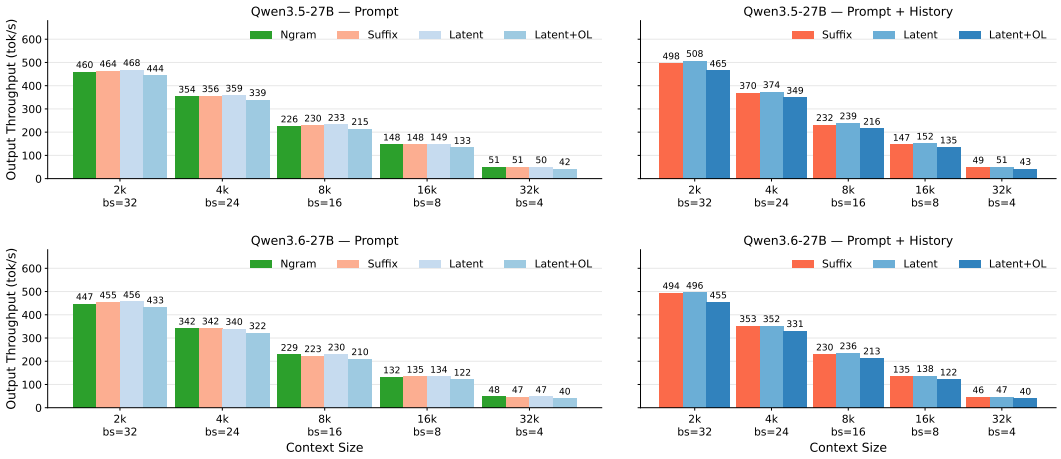


Figure 8: **Throughput measurement on speedbench (Qwen3.5-9B, B200)**